

Jürgen Beisert

# Kritzel Manual

This manual covers hard- and software



1002 2009-01-05 14:27:26 +0100 (Mon, 05 Jan 2009)

# Contents

<b>I</b>	<b>User's Manual for JB Kritzel</b>	<b>4</b>
<b>1</b>	<b>User's Manual</b>	<b>5</b>
1.1	What is Kritzel?	5
1.2	Device Controls	6
1.3	Man Page	7
1.4	Installation	7
1.5	Preparation on the Host's Side	8
1.6	How to measure	8
1.6.1	Physical Connections	8
1.6.2	Start Measuring	9
1.6.3	What a Reporting Interval Means	10
1.7	Probe Protection	11
1.8	How to Stimulate	11
1.9	Data Processing	12
1.9.1	Simple Data display	12
1.9.2	Check Data Consistency	14
1.9.3	Display specific Reports	14
1.9.4	Creating a Histogram	14
1.9.5	Visualizing with <i>gtkwave</i>	15
<b>II</b>	<b>The soft Side of this Project</b>	<b>17</b>
<b>2</b>	<b>Some Notes first</b>	<b>18</b>
<b>3</b>	<b>Getting a working Environment</b>	<b>19</b>
3.1	Download Software Components	19
3.2	PTXdists Installation	19
3.2.1	Main parts of PTXdists	19
3.2.2	Extracting the Sources	20
3.2.3	Prerequisites	21
3.2.4	Configuring PTXdists	22
3.3	Toolchains	23
3.3.1	Abstract	23
3.3.2	Using Existing Toolchains	24
3.3.3	Building a Toolchain	25
3.3.4	Freezing the Toolchain	26
<b>4</b>	<b>Building Description</b>	<b>27</b>

4.1 Prepare and Build . . . . .	27
<b>III The hard Side of this Project</b>	<b>28</b>
<b>5 Preparations and preconditions</b>	<b>29</b>
5.1 Required Skills . . . . .	29
5.2 Required Tools . . . . .	29
<b>6 Bill of Material</b>	<b>31</b>
6.1 Simple Bill for Assembling . . . . .	31
6.2 Ordering Information . . . . .	32
<b>7 Assembly</b>	<b>34</b>
7.1 The PCB from the CAD system . . . . .	34
7.2 The PCB from the Manufacturer . . . . .	34
7.3 Check the plain PCB . . . . .	34
7.4 Soldering . . . . .	35
7.4.1 Step 1: The small devices . . . . .	35
7.4.2 Checking Step 1 . . . . .	35
7.4.3 Step 2: The Processor . . . . .	36
7.4.4 Checking Step 2 . . . . .	37
7.4.5 Step 3: The USB device . . . . .	39
7.4.6 Checking Step 3 . . . . .	40
7.4.7 Step 4: The Input/Output Drivers . . . . .	41
7.4.8 Checking Step 4 . . . . .	41
7.4.9 Finishing the housing . . . . .	42
7.5 Programming the FTDI's EEPROM . . . . .	45
7.6 Front Panel . . . . .	50
7.6.1 Panel's Dimensions . . . . .	50
<b>8 Layout Component Placement</b>	<b>51</b>
8.1 Top Side Component Placement . . . . .	51
8.2 Bottom Side Component Placement . . . . .	52
<b>9 Schematics</b>	<b>53</b>
<b>10 Help</b>	<b>58</b>

## **Part I**

# **User's Manual for JB Kritzel**

# 1 User's Manual

This chapter should help any new user to get his/her FirstGeneration Kritzel to run.



All the programs mentioned here in this manual are built from source. Refer chapter II how to do so.

## 1.1 What is Kritzel?

Kritzel is a simple scanner for general purpose measuring of digital signals up to about 250 kHz. It is host based, so any data processing occurs at the host side. Kritzel only reports what is happening on the probe leads and when.

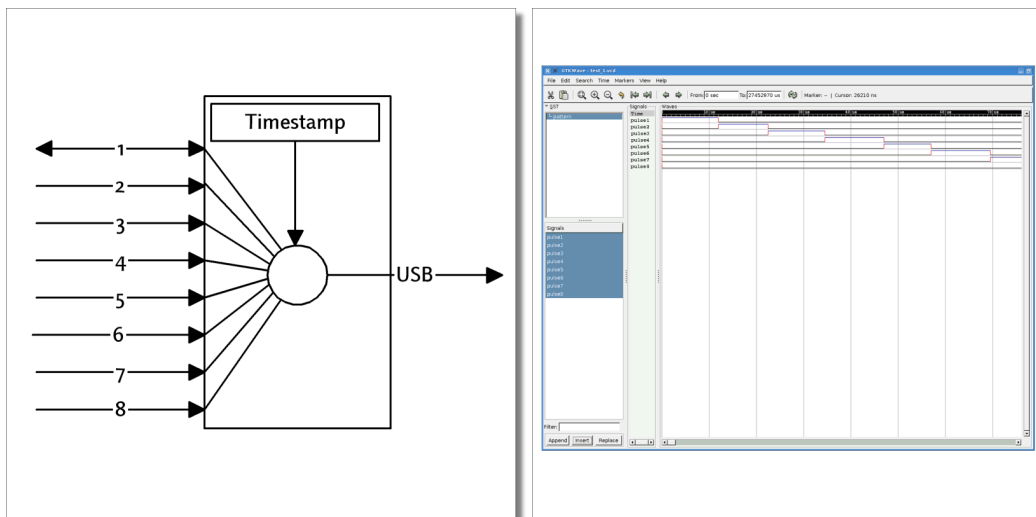


Figure 1.1: Building Blocks / Visualizing with gtkwave

Other features:

- Fullspeed USB
- USB powered, no additional power supply required
- 7 input-only probes, 1 bidirectional probe
- PWM for stimulus application on external devices under test

- Two 3.3 V level probes (5 *volt* tolerant), six 5 *volt* level probes

Kritzel supports the VCD (Value Change Dump) data format, so the results can be processed and visualized with gtkwave (<http://home.nc.rr.com/gtkwave/>) or similar applications.

## 1.2 Device Controls

Kritzel can be controlled by the host, but also a measurement can be started and stopped locally at the device. A few LEDs are showing the current state of the device. Figure 1.2 shows the locations of these control items.

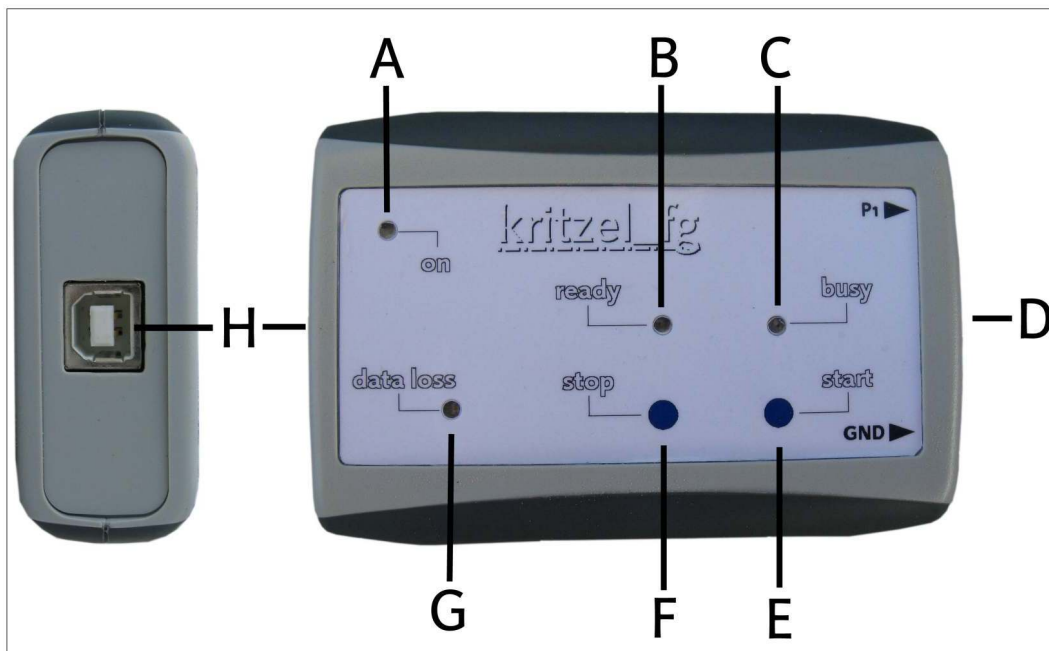


Figure 1.2: Controls and Connectors

- A On LED. Lights green, when USB power is active
- B Ready LED. Lights green, when device is ready to be used and the start button (E) and stop buttons (F) can be used
- C Busy LED. Lights red when a measurement is running
- D 8 Probe connectors plus one ground connector (not shown here)
- E Start button. Pressing this button starts a measurement in interactive mode
- F Stop button. Pressing this button stops a measurement in interactive mode
- G Data Loss LED. Lights yellow, when more events happen than reports can be transferred through the USB line
- H USB connector

## 1.3 Man Page

Run the program with:

```
~# kritzel [options] <device name>
```

Options are:

- s <value> Define scan interval (unit: ns). Default is 10 000 000 = 10 ms.
- i <value> Define stimulus interval (unit: ns). Disabled by default.
- a <value> Define stimulus active time (unit: ns). Default is 1 000 000 = 1 ms.
- l <value> Define stimulus active level (0|1). Default is 1 = active high.
- o <basename> Use this basename for all output files. Default is *test\_file*.
- t <text> Define the measurement title. Default is *'Measuring'*.
- r Start scan immediately, else wait for the scanner's local start action.
- f <format> Select the data format on **stdout**. <format> could be:
  - *'krt'* Kritzel's native format.
  - *'vcd'* Industrial VCD format.
- <no> <name> Define a probe name (<no> from 1 to 8).

<device name> depends on the device node name *udev* gives your device. If it is the only serial device, usually it will be */dev/ttyUSB0*.

## 1.4 Installation

To install the package, ensure *libz* is already installed on your system.

```
~# tar xf kritzel-1.0.0.tar.bz2
~# cd kritzel-1.0.0
~/kritzel-1.0.0 # configure
~/kritzel-1.0.0 # make
~/kritzel-1.0.0 # sudo make install
```

Most things are done automatically by *configure*, but there are some parameters that can control how to build the kritzel executable.

- enable-debug** Be more noisy and do more runtime checking. Enabled by default. You should disable this feature when you only use this program.
- enable-krt\_format** This is the native format kritzel uses when it outputs data to stdout. Don't mistake it with the compressed data format. This is valid only for stdout.
- enable-vcd\_format** This is a industrial format used by other tools and also by *gtkwave*. Support of this format is enabled by default.

## 1.5 Preparation on the Host's Side

The FirstGeneration Kritzel works with a parallel to USB converter from the vendor FTDI. The device is an FT245BM (refer to <http://www.ftdichip.com/>). This kind of device is supported since the Linux kernel 2.4 days. When it integrates itself into the system, it emulates a simple serial connection, so every tool that supports tty communications can work with it.

When running udev on the host, it creates for each connected FT245BM a device node in this form:

`/dev/ttyUSBx`

The 'x' part in this device node name will be a number starting at 0. It's possible to connect more than one device at the same time.



The FTDI device uses a 16 ms timeout value as default to transfer data from its FIFO to the host, if there are less than 64 bytes in it. To speed things up this value should be decreased to 1 ms. This can be done with a:

```
$ echo 1 > /sys/class/tty/ttyUSB?/device/latency_timer
```

Note: Replace the ? with the correct number udev has given your Kritzel.

---

## 1.6 How to measure

Up to eight probes can be used to measure digital signals. Two types of levels are currently supported:

- Probes #1 and #2 are using input stages with 3.3 V power supply. So they are reporting levels below 1.0 V as 0 and levels above 1.5 V as 1.
- Probes #3 to #8 are using input stages with 5.0 V power supply. They are reporting levels below 1.2 V as 0 and levels above 2.1 V as 1.

Connect as much probes as you want to observe signals and – don't forget – the GND probe.



Do not connect the GND probe to a low impedance source other than ground. This may destroy your target, Kritzel and your host immediately.

---

Connect Kritzel through the USB to your host. A device like `/dev/ttyUSB0` should now show up.

### 1.6.1 Physical Connections

If you prepare the probe cables in the same way as me, you can do the connections in two ways: Directly through header pins or through any kind of clip contact. The micro clip contacts are very cool, but also expensive.

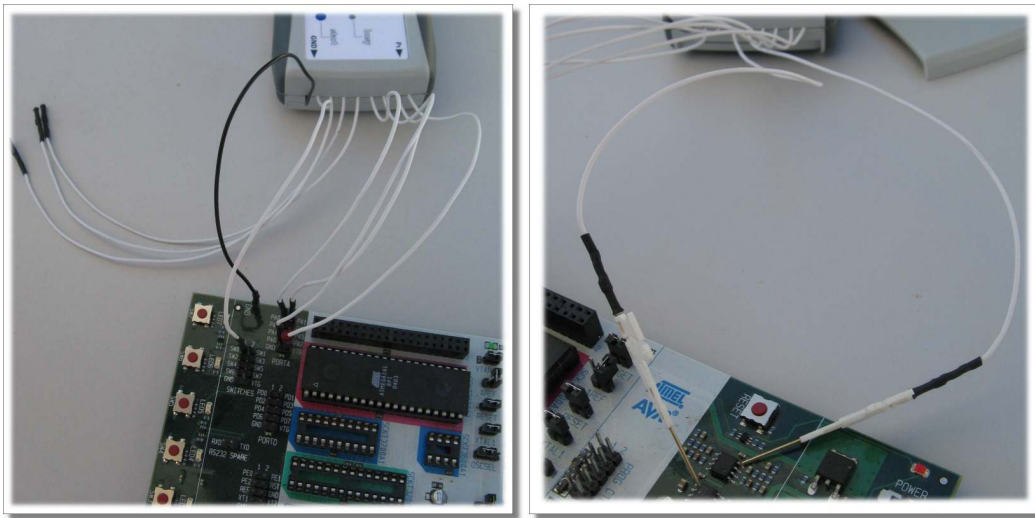


Figure 1.3: Direct connection through header pins (left) and through micro clips (right)

### 1.6.2 Start Measuring

To start a measurement, run at least the `kritzel` program on your host. It needs only one parameter (all other settings are using default values in this case): The device node to get a connection to the Kritzel system.

```
# kritzel /dev/ttyUSB0
```

This will use a reporting cycle of 10 ms and stores all data into file `test_file.krt` and in addition some human readable info about this measurement in `test_file.info`.



The `kritzel` application doesn't do any data processing. It only collects all events reported by the scanner and stores them to a file. For any further data processing other applications are required that can work on the stored data.

---

To select another reporting interval than the default, extend the command line options with `-s`. To get a reporting interval of 4  $\mu$ s change the command line to:

```
# kritzel -s 4000 /dev/ttyUSB0
```

The reporting interval depends on hardware capabilities. Not every interval is possible. Kritzel selects the nearest possible interval automatically.

For the FirstGeneration Kritzel these intervals are possible:

- 500 ns
- 4  $\mu$ s
- 16  $\mu$ s
- 64  $\mu$ s

Note: The 500 ns interval is an internal interval only. This means Kritzel can detect events with this timing resolution. But after an event detection the firmware needs 2  $\mu$ s to generate a report and forward it to the USB FIFO. So the maximum external frequency is about 500 kHz. Note also, Kritzel cannot generate reports permanently at this rate. Because each report contains 4 bytes, this would result in a 2 MiB/s data rate, which a full speed USB device cannot handle.

This means a short burst of a 500 kHz signal Kritzel can handle until the FIFO is full. The average external frequency Kritzel can handle permanently is about 250 kHz.

### 1.6.3 What a Reporting Interval Means

Each reporting interval has a timestamp and all events within this interval are collected and reported at the end of that reporting interval (refer to figure 1.4). 'All events' means all events on all signals and it includes if one or more signals change their level more than once in the same reporting interval. If the latter case happens, Kritzel uses an "event loss" mark for these signals in this report.

If there was no event within a single reporting interval, there will also be no report. This keeps the data amount small in the cases where events happen at a low rate.

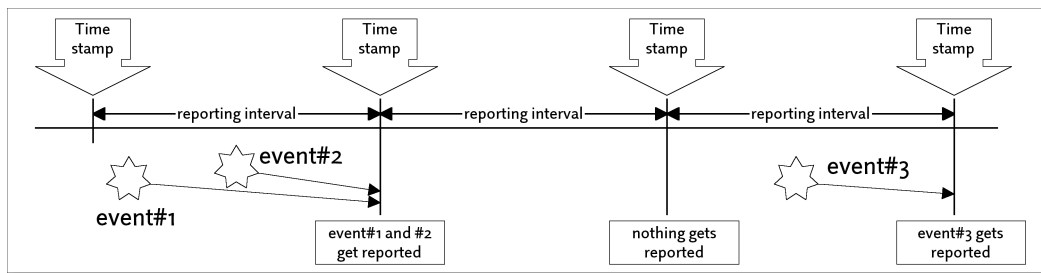


Figure 1.4: Reporting Intervals

Note: Due to hardware restrictions the reporting interval 500 ns will generate a report whenever a single event was detected (no collection). The hardware is not fast enough to detect more than one event in this short interval.

To distinguish all measurements later on, you can give each a title and a name for each probe. Other data processing applications can use this information.

```
# kritzel -s 4000 -t "This is a specific title" /dev/ttyUSB0
```

There is no length restriction for the title. You only should avoid characters like '@' and '\$'. If you use them, they will get replaced by '\_' and '#'. This restriction exists, as these characters are used to separate fields in Kritzel's data files.

To define a name for each probed signal you can extend the command line like this:

```
# kritzel -s 4000 -1 trigger -2 answer -8 interrupt /dev/ttyUSB0
```

This will assign the name 'trigger' to probe #1, 'answer' to probe #2 and 'interrupt' to probe #8. Probes #3 ... #7 are still using their default names.

There is no length restriction for each name, but you should avoid the same characters in the name as for the measurement title.

Each run of `kritzel` will store the event data into `test_file.krt` and `test_file.info`. If you are using the `-o` command line option, you can select a different basename than `test_file`. `kritzel` will extend this basename with `.krt` and `.info` by itself!

## 1.7 Probe Protection

All inputs are protected against electrostatic discharge (ESD) with a serial resistor and clamping diodes (refer to figure 1.5). This will also clamp 5 V input signals at probe #1 and probe #2. But avoid to connect any low impedance source to any of the probe inputs that will force the clamping diodes to do their work. This will destroy at least the serial resistors. If the input voltage is above 7 V, all active devices will be damaged, and as the last member in this chain your host computer!

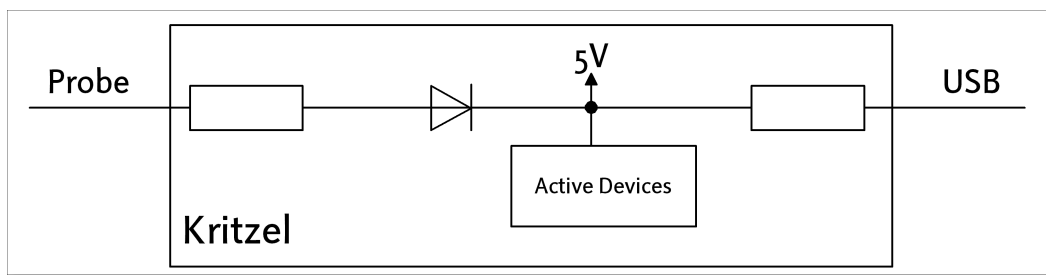


Figure 1.5: Protection Scheme

## 1.8 How to Stimulate

When you set up the stimulus generation, Kritzel will enable its output buffer on probe #1. Note: Currently only probe #1 can be a stimulus output. All other probes are input only.

As probe #1 continues to be an input too, the stimulus data will be part of the data set. If there is something to measure in relation to the stimulus, always select probe #1 as one of the edges to be used for the calculation.

Note: Stimulus generation depends on the capabilities of the underlying hardware. The FirstGeneration Kritzel uses an internal 8 bit timer generating the output stimulus. So the stimulus interval and the active time depend on each other.

You can define the stimulus interval with the `-i` command line parameter, the active time with `-a` and the active level with `-l`.

The stimulus signal is a simple PWM signal, generated by an 8 bit counter with compare capability. To setup the nearest value to the given one, Kritzel's firmware first tries to find the best stimulus interval. Due to only a few internal clock dividers, the intervals are limited to (granularity in parentheses):

- 16  $\mu$ s (62.5 ns)
- 128  $\mu$ s (500 ns)
- 512  $\mu$ s (2  $\mu$ s)

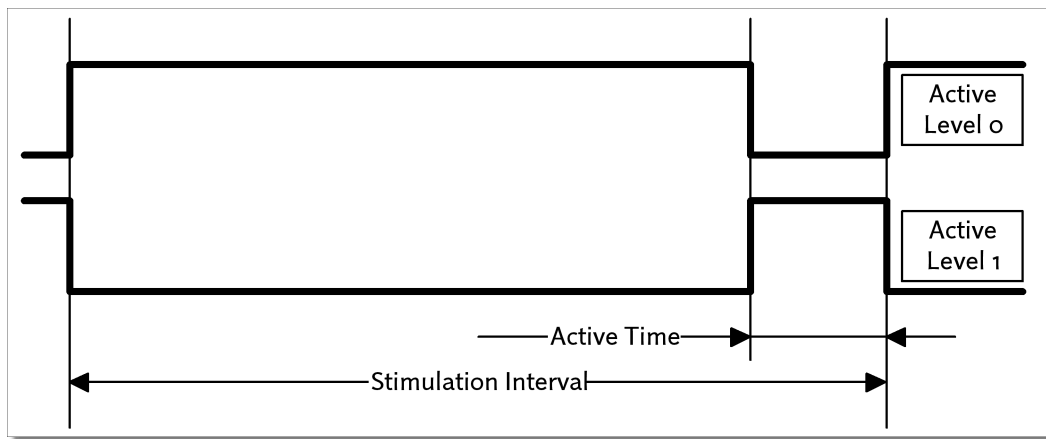


Figure 1.6: Stimulus Generation Overview

- 1.024 ms (4  $\mu$ s)
- 2.048 ms (8  $\mu$ s)
- 4.096 ms (16  $\mu$ s)
- 16.384 ms (64  $\mu$ s)

After selecting the best stimulus interval the firmware searches for the nearest possible active time. Refer to the values in parentheses for the granularity in each stimulus interval. Based on these values the firmware will calculate the possible active time within the stimulus interval.

Example: For the 16.384 ms stimulus interval it means the shortest possible active time will be 64  $\mu$ s. Longer active times will be always multiples of 64  $\mu$ s.

## 1.9 Data Processing

The kritzel main program generates up to two data streams from the reports the scanner sends. One stream is a compressed one and always generated. The filename is `test_file.krt` (or a different basename, when using the command line option `-o`).

A second stream will be generated if kritzel found its `stdout` redirected. This stream can be used to work live on the data. All other data processing programs in this archive can work with the compressed file or the redirected stream.

### 1.9.1 Simple Data display

`kritzel_raw` displays the scanner data in a primitive way. You can run this program on live data or on an already stored `*.krt` file.

Display live data:

```
# kritzel /dev/ttyUSB0 | kritzel_raw
```

Display stored data:

```
# kritzal_raw <file_name.krt>
```

kritzal\_raw has an useful option when handling large data sets. It can start to display with an data set offset. This option is:

```
-n <value>
```

Value is a simple number. When given, kritzal\_raw skips the first number reports before starting to display.

How to read its output?

```
[...]  
11C: |#| | | | | (+24)  
131: ||#| | | | (+21)  
149: |||#| | | (+24)  
162: ||||#| | | (+25)  
[...]  
F958: || | | | |#| (+24)  
F96D: || | | | | |# (+21)  
F985: || | | | | | | (+24)  
    1: || | | | | | | (+1660)  
157: #| | | | | | | (+342)  
[...]
```

First column is the timestamp. It's displayed in hex format. It's always 16 bit wide and overflows at 0xFFFF. The next columns are showing the states of the 8 probe signals at this timestamp (from left to right: Probe #1 to #8). The character 'l' represents a 0 (low), the '#' represents a 1 (high).

The third column in the example above is empty (description see below).

The fourth column shows the timestamp difference to the previous report. Note: The timestamp is always in relation to the selected time resolution while scanning! So if the timestamp value advances by one, this could mean 500 ns, 4  $\mu$ s and so on.

Lets explain the second row in the example above:

The report was generated at the timestamp 0x131. It happens 21 counts later than the previous report. Probe #1, #4, #5, #6, #7 and #8 do not change their state. Probe #2 changes from 1 to 0, while probe #3 changes from 0 to 1.

Sometimes a row contains the third column mentioned above:

```
[...]  
1B5: || | | |#| | | (+24)  
1CA: || | | |#| | | (+21)  
1E2: || | | | |#| .....!.. (+24)  
1FA: || | | | | |# (+24)  
20F: || | | | | | | (+21)  
[...]
```

At timestamp 0x1E2 the scanner has detected more than one event on probe #6. This means the level on this probe changed at least two times. In this example at least from 1 (at timestamp 0x1CA) to 0, back to 1 and again to 0 (the final state at timestamp 0x1E2).

There are different reasons for this to occur:

- The report interval is too long for the frequency on the probe lines.
- The USB FIFO was full. While the firmware waits for new space in the FIFO it continues to collect events.

### 1.9.2 Check Data Consistency

This tool is only useful for the stored data set. It was very helpful during the development of this project. It could become helpful again, when someone has trouble while using programs of this project.

```
# kritznel_check <file_name.krt>
Reading data from file test_1.krt
Data file contains 214661 datasets
No obvious errors detected
```

### 1.9.3 Display specific Reports

This tool was developed to check the realtime capabilities of the current Linux RT Preempt patch.

To find specific reports showing (for example) a timing violation `kritznel_select` can be used. With the help of `kritznel_select` you can setup two event sources and trigger (results into raw data display) if these events violates a given timing.

Run `kritznel_select --help` for the possible options.

### 1.9.4 Creating a Histogram

This tool was developed to check the realtime capabilities of the current Linux RT Preempt patch.

While the scanner generates a stimulus at a specific rate for a target it scans the target's answers on a second probe. If the stimulus is connected to an interrupt input, the target can answer interrupt recognition on the second probe. Based on this dataset you can create a histogram to measure the realtime capabilities of the target while running various loads on it.

The tool `kritznel_histogram` creates a histogram in ASCII format, that can be used with *gnu plot* to get nice graphical histograms.

Example:

- The scanner outputs a 1 kHz signal on probe #1. This signal is fed into an interrupt input of our target. The interrupt input is active low.
- The target's interrupt routine outputs an answer on a separate GPIO. It toggles the GPIO whenever it received the interrupt and entered the handler routine. This GPIO is connected to the scanner's probe #2.

Start the test with:

```
# kritzal -s 500 -i 1000000 -a 200000 -l 0 -1 stimul -2 answer -t "realtime" /dev/ttyUSB0
```

Now run various loads on your target to measure its realtime capabilities.

When the measurement is done, you can build the histogram for this test. We are interested in the timing starting with the falling edge of probe #1 (interrupt) and ending with both edge of the answer signal on probe #2.

```
# kritzal_histogram -b 1- -e 2 test_file.krt
```

```
# Histogram
# realtime
# Reporting interval: 500 ns
# From signal stimul (1) falling edge to signal answer (2) both edges
# Data sets over all: 214662. Counted events: 23733. Erroneous events: 0.
# First collumn: Time in [ns], second collumn: Count
  8000      112
  9500      291
 10000      301
 10500      376
 11500      198
 12500      191
 13000      154
 15000       85
 17500       69
 20500       12
 24500        2
```

The worst case in this example is about 25  $\mu$ s. The graphical histogram can be found in figure 1.7.

### 1.9.5 Visualizing with *gtkwave*

The application *gtkwave* can be used to visualize the data set in a more convenient way than *kritzal\_raw* can do it.

To get a dataset in a fileformat that *gtkwave* can handle, use the '-f vcd' option and redirection when running the measurement.

```
# kritzal -s 500 -f vcd > test.vcd
```

Note: The file `test.vcd` will grow quickly and can be very huge!

To visualize this dataset run *gtkwave* and simply load the `test.vcd` file.

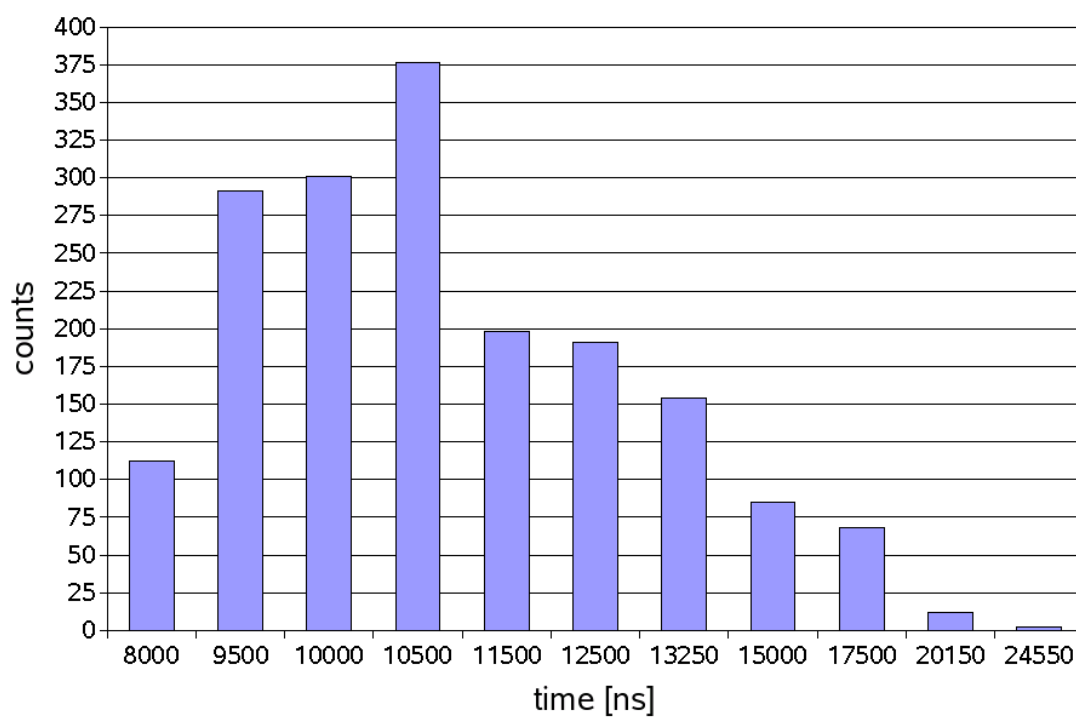


Figure 1.7: Histogram

## **Part II**

# **The soft Side of this Project**

## 2 Some Notes first

All the needed software components to build and run Kritzel will be made by an automated build system. This build system is called PTXdist and its regular purpose is to build root filesystems for Linux based targets.

This project is not like all the other PTXdist based projects. The main difference is the lack of a Linux based system on target's side. Instead the target is a small microcontroller without the need for a complex operating system.

But PTXdist is a generic build system to simplify building processes of any kind of software. So its also used here to provide any user with all the programs he or she will need to setup and run this piece of hardware.

To create the binaries for the Kritzel scanner a *Cross Toolchain* for the 8 bit AVR architecture is required. This toolchain will also be built by PTXdist, so no special knowledge about toolchain building is needed.

Before you begin to assemble anything, build the whole software first. It includes some testprograms you will need to check if an assemble step was successfull or not. They should simplify things in case of trouble.

Note: Everything in this project comes in source format, there is nothing binary only in it. But not all needed source parts are part of this archive. They will be downloaded automatically on request from various internet locations.

## 3 Getting a working Environment

This chapter describes how to install PTXdist and building a toolchain. PTXdist and the toolchain are the major preconditions to build anything other. You can pass PTXdist and do it by your own. There is nothing esoteric in my source code. Just plain Makefiles and/or autotoolized programs. But then you need at least a known to work AVR toolchain.

But please note: If you do it by your own with your own toolchain I cannot answer questions if something went wrong.

That's the job of PTXdist: It ensures that your binary results are exactly the same than mine.

### 3.1 Download Software Components

In order to follow this manual, some software archives are needed. There are several possibilities how to get these: either as part of an evaluation board package or by downloading them from the Pengutronix web site.

The central place for OSELAS related documentation is <http://www.oselas.com>. This website provides all required packages and documentation (at least for software components which are available to the public).

To build OSELAS.BSP-JB-Kritzel-trunk, the following archives have to be available on the development host:

- ptxdist-1.99.10.tgz
- ptxdist-1.99.10-patches.tgz
- OSELAS.BSP-JB-Kritzel-trunk.tar.gz
- OSELAS.Toolchain-1.99.2.tar.bz2

If they are not available on the development system yet, it is necessary to get them.

### 3.2 PTXdist Installation

The PTXdist build system can be used to build a root filesystem for embedded Linux devices. In order to start development with PTXdist it is necessary to install the software on the development system.

This chapter provides information about how to install and configure PTXdist on the development host.

#### 3.2.1 Main parts of PTXdist

The most important software component which is necessary to build an OSELAS.BSP( ) board support package is the PTXdist tool. So before starting any work we'll have to install PTXdist on the development host.

PTXdist consists of the following parts:

**The ptxdist Program:** ptxdist is installed on the development host during the installation process. ptxdist is called to trigger any action, like building a software packet, cleaning up the tree etc. Usually the ptxdist program is used in a *workspace* directory, which contains all project relevant files.

**A Configuration System:** The config system is used to customize a *configuration*, which contains information about which packages have to be built and which options are selected.

**Patches:** Due to the fact that some upstream packages are not bug free – especially with regard to cross compilation – it is often necessary to patch the original software. PTXdist contains a mechanism to automatically apply patches to packages. The patches are bundled into a separate archive. Nevertheless, they are necessary to build a working system.

**Package Descriptions:** For each software component there is a "recipe" file, specifying which actions have to be done to prepare and compile the software. Additionally, packages contain their configuration snippet for the config system.

**Toolchains:** PTXdist does not come with a pre-built binary toolchain. Nevertheless, PTXdist itself is able to build toolchains, which are provided by the OSELAS.Toolchain() project. More in-deep information about the OSELAS.Toolchain() project can be found here: [http://www.pengutronix.de/oselas/toolchain/index\\_en.html](http://www.pengutronix.de/oselas/toolchain/index_en.html)

**Board Support Package** This is an optional component, mostly shipped aside with a piece of hardware. There are various BSP available, some are generic, some are intended for a specific hardware.

### 3.2.2 Extracting the Sources

To install PTXdist, at least two archives have to be extracted:

**ptxdist-1.99.10.tgz** The PTXdist software itself.

**ptxdist-1.99.10-patches.tgz** All patches against upstream software packets (known as the 'patch repository').

**ptxdist-1.99.10-projects.tgz** Generic projects (optional), can be used as a starting point for self-built projects.

The PTXdist and patches packets have to be extracted into some temporary directory in order to be built before the installation, for example the `local/` directory in the user's home. If this directory does not exist, we have to create it and change into it:

```
~# cd
~# mkdir local
~# cd local
```

Next steps are to extract the archives:

```
~/local# tar -zxf ptxdist-1.99.10.tgz
~/local# tar -zxf ptxdist-1.99.10-patches.tgz
```

and if required the generic projects:

```
~/local# tar -zxf ptxdist-1.99.10-projects.tgz
```

If everything goes well, we now have a PTXdist-1.99.10 directory, so we can change into it:

```
~/local# cd ptxdist-1.99.10
~/local/ptxdist-1.99.10# ls -l
```

```
total 455
drwxr-xr-x  2 jb users  1024 29. Sep 17:32 autoconf/
-rwxr-xr-x  1 jb users    28 29. Sep 17:32 autogen.sh*
drwxr-xr-x  2 jb users  1024 29. Sep 17:32 bin/
-rw-r--r--  1 jb users 115470 29. Sep 17:32 ChangeLog
drwxr-xr-x  5 jb users  1024 29. Sep 17:32 config/
-rwxr-xr-x  1 jb users 222451 29. Sep 17:34 configure*
-rw-r--r--  1 jb users  11445 29. Sep 17:34 configure.ac
-rw-r--r--  1 jb users  18361 29. Sep 17:32 COPYING
-rw-r--r--  1 jb users   3499 29. Sep 17:32 CREDITS
drwxr-xr-x  2 jb users  1024 29. Sep 17:30 debian/
drwxr-xr-x  2 jb users  1024 29. Sep 17:32 Documentation/
drwxr-xr-x  7 jb users  1024 29. Sep 17:32 generic/
-rw-r--r--  1 jb users    58 29. Sep 17:32 INSTALL
-rw-r--r--  1 jb users  2150 29. Sep 17:32 Makefile.in
drwxr-xr-x 159 jb users  4096 29. Sep 17:31 patches/
drwxr-xr-x  2 jb users  1024 29. Sep 17:32 platforms/
drwxr-xr-x  4 jb users  1024 29. Sep 17:30 plugins/
-rw-r--r--  1 jb users  4091 29. Sep 17:32 README
-rw-r--r--  1 jb users   691 29. Sep 17:32 REVISION_POLICY
drwxr-xr-x  6 jb users 28672 29. Sep 17:32 rules/
drwxr-xr-x  6 jb users  1024 29. Sep 17:30 scripts/
drwxr-xr-x  2 jb users  1024 29. Sep 17:32 tests/
-rw-r--r--  1 jb users 33418 29. Sep 17:32 TODO
```

### 3.2.3 Prerequisites

Before PTXdist can be installed it has to be checked if all necessary programs are installed on the development host. The configure script will stop if it discovers that something is missing.

The PTXdist installation is based on GNU autotools, so the first thing to be done now is to configure the packet:

```
~/local/ptxdist-1.99.10# ./configure
```

This will check your system for required components PTXdist relies on. If all required components are found the output ends with:

```
[...]
checking whether /usr/bin/patch will work... yes

configure: creating ./config.status
config.status: creating Makefile
config.status: creating scripts/ptxdist_version.sh
config.status: creating rules/ptxdist-version.in
```

```
ptxdist version 1.99.10 configured.  
Using '/usr/local' for installation prefix.
```

```
Report bugs to ptxdist@pengutronix.de
```

Without further arguments PTXdist is configured to be installed into `/usr/local`, which is the standard location for user installed programs. To change the installation path to anything non-standard, we use the `--prefix` argument to the `configure` script. The `--help` option offers more information about what else can be changed for the installation process.

The installation paths are configured in a way that several PTXdist versions can be installed in parallel. So if an old version of PTXdist is already installed there is no need to remove it.

One of the most important tasks for the `configure` script is to find out if all the programs PTXdist depends on are already present on the development host. The script will stop with an error message in case something is missing. If this happens, the missing tools have to be installed from the distribution before re-running the `configure` script.

When the `configure` script is finished successfully, we can now run

```
~/local/ptxdist-1.99.10# make
```

All program parts are being compiled, and if there are no errors we can now install PTXdist into its final location. In order to write to `/usr/local`, this step has to be performed as user `root`:

```
~/local/ptxdist-1.99.10# sudo make install  
[enter root password]  
[...]
```

If we don't have root access to the machine it is also possible to install into some other directory with the `--prefix` option. We need to take care that the `bin/` directory below the new installation dir is added to our `$PATH` environment variable (for example by exporting it in `~/ .bashrc`).

The installation is now done, so the temporary folder may now be removed:

```
~/local/ptxdist-1.99.10# cd  
~# rm -fr local
```

#### 3.2.4 Configuring PTXdist

When using PTXdist for the first time, some setup properties have to be configured. Two settings are the most important ones: Where to store the source packages and if a proxy must be used to gain access to the world wide web.

Run PTXdist's setup:

```
~# ptxdist setup
```

Due to PTXdist is working with sources only, it needs various source archives from the world wide web. If these archives are not present on our host, PTXdist starts the `wget` command to download them on demand.

### 3.2.4.1 Proxy Setup

To do so, an internet access is required. If this access is managed by a proxy `wget` command must be advised to use it. PTXdist can be configured to advice the `wget` command automatically: Navigate to entry *Proxies* and enter the required addresses and ports to access the proxy in the form:

`<protocol>://<address>:<port>`

### 3.2.4.2 Source Archive Location

Whenever PTXdist downloads source archives it stores it project locally. If we are working with more than one project, every project would download its own required archives. To share all source archives between all projects PTXdist can be configured to use only one archive directory for all projects it handles: Navigate to menu entry *Source Directory* and enter the path to the directory where PTXdist should store archives to share between projects.

### 3.2.4.3 Generic Project Location

If we already installed the generic projects we should also configure PTXdist to know this location. If we already did so, we can use the command `ptxdist projects` to get a list of available projects and `ptxdist clone` to get a local working copy of a shared generic project.

Navigate to menu entry *Project Searchpath* and enter the path to projects that can be used in such a way. Here we can configure more than one path, each part can be delimited by a colon. For example for PTXdist's generic projects and our own previous projects like this:

```
/usr/local/lib/ptxdist-1.99.10/projects:/office/my_projects/ptxdist
```

Leave the menu and store the configuration. PTXdist is now ready for use.

## 3.3 Toolchains

### 3.3.1 Abstract

Before we can start building our first userland we need a cross toolchain. On Linux, toolchains are no monolithic beasts. Most parts of what we need to cross compile code for the embedded target comes from the *GNU Compiler Collection*, `gcc`. The `gcc` packet includes the compiler frontend, `gcc`, plus several backend tools (`cc1`, `g++`, `ld` etc.) which actually perform the different stages of the compile process. `gcc` does not contain the assembler, so we also need the *GNU Binutils package* which provides lowlevel stuff.

Cross compilers and tools are usually named like the corresponding host tool, but with a prefix – the *GNU target*. For example, the cross compilers for ARM and powerpc may look like

- `arm-softfloat-linux-gnu-gcc`
- `powerpc-unknown-linux-gnu-gcc`

With these compiler frontends we can convert e.g. a C program into binary code for specific machines. So for example if a C program is to be compiled natively, it works like this:

```
~# gcc test.c -o test
```

To build the same binary for the ARM architecture we have to use the cross compiler instead of the native one:

```
~# arm-softfloat-linux-gnu-gcc test.c -o test
```

Also part of what we consider to be the "toolchain" is the runtime library (libc, dynamic linker). All programs running on the embedded system are linked against the libc, which also offers the interface from user space functions to the kernel.

The compiler and libc are very tightly coupled components: the second stage compiler, which is used to build normal user space code, is being built against the libc itself. For example, if the target does not contain a hardware floating point unit, but the toolchain generates floating point code, it will fail. This is also the case when the toolchain builds code for i686 CPUs, whereas the target is i586.

So in order to make things working consistently it is necessary that the runtime libc is identical with the libc the compiler was built against.

PTXdist doesn't contain a pre-built binary toolchain. Remember that it's not a distribution but a development tool. But it can be used to build a toolchain for our target. Building the toolchain usually has only to be done once. It may be a good idea to do that over night, because it may take several hours, depending on the target architecture and development host power.

#### 3.3.2 Using Existing Toolchains

If a toolchain is already installed which is known to be working, the toolchain building step with PTXdist may be omitted.



The OSELAS.BoardSupport() Packages shipped for PTXdist have been tested with the OSELAS.Toolchains() built with the same PTXdist version. So if an external toolchain is being used which isn't known to be stable, a target may fail. Note that not all compiler versions and combinations work properly in a cross environment.

---

Every OSELAS.BoardSupport() Package checks for its OSELAS.Toolchain it's tested against, so using a different toolchain vendor requires an additional step:

Open the OSELAS.BoardSupport() Package menu with:

```
~# ptxdist menuconfig
```

and navigate to PTXdist Config, Architecture and Check for specific toolchain vendor. Clear this entry to disable the toolchain vendor check.

### 3.3.3 Building a Toolchain

PTXdist-1.0.2 handles toolchain building as a simple project, like all other projects, too. So we can download the OSELAS.Toolchain bundle and build the required toolchain for the OSELAS.BoardSupport() Package.

A PTXdist project generally allows to build into some project defined directory; all OSELAS.Toolchain projects that come with PTXdist are configured to use the standard installation paths mentioned below.

All OSELAS.Toolchain projects install their result into `/opt/OSELAS.Toolchain-1.99.2/`.



---

Usually the `/opt` directory is not world writable. So in order to build our OSELAS.Toolchain into that directory we need to use a root account to change the permissions so that the user can write (`mkdir /opt/OSELAS.Toolchain-1.99.2`;  
; `chown <username> /opt/OSELAS.Toolchain-1.99.2`; `chmod a+rw`  
`/opt/OSELAS.Toolchain-1.99.2`).

---

We recommend to keep this installation path as PTXdist expects the toolchains at `/opt`. Whenever we go to select a platform in a project, PTXdist tries to find the right toolchain from data read from the platform configuration settings and a toolchain at `/opt` that matches to these settings. But that's for our convenience only. If we decide to install the toolchains at a different location, we still can use the *toolchain* option to define the toolchain to be used on a per project base.

#### 3.3.3.1 Building the OSELAS.Toolchain for OSELAS.BSP-JB-Kritzel-trunk

To compile and install an OSELAS.Toolchain we have to extract the OSELAS.Toolchain archive, change into the new folder, configure the compiler in question and start the build.

The required compiler to build the OSELAS.BSP-JB-Kritzel-trunk board support package is

```
avr/avr_gcc-4.3.2_libc-1.6.2_binutils-2.19
```

So the steps to build this toolchain are:

```
~# tar xf OSELAS.Toolchain-1.99.2.tar.bz2
~# cd OSELAS.Toolchain-1.99.2
~/OSELAS.Toolchain-1.99.2# ptxdist select
    ptxconfigs/avr/avr_gcc-4.3.2_libc-1.6.2_binutils-2.19.ptxconfig
~/OSELAS.Toolchain-1.99.2# ptxdist go
```

At this stage we have to go to our boss and tell him that it's probably time to go home for the day. Even on reasonably fast machines the time to build an OSELAS.Toolchain is something like around 30 minutes up to a few hours.

Measured times on different machines:

- Single Pentium 2.5 GHz, 2 GiB RAM: about 2 hours
- Dual Athlon 2.1 GHz, 2 GiB RAM: about 1 hour 20 minutes
- Dual Quad-Core-Pentium 1.8 GHz, 8 GiB RAM: about 25 minutes

Another possibility is to read the next chapters of this manual, to find out how to start a new project.

When the OSELAS.Toolchain project build is finished, PTXdist is ready for prime time and we can continue with our first project.

#### 3.3.4 Freezing the Toolchain

As we build and install this toolchain with regular user permissions we should modify the permissions as a last step to avoid any later manipulation. To do so we could set all toolchain files to read only or change recursively the owner of the whole installation to user root.

This is an important step for reliability. Do not omit it!

##### 3.3.4.1 Building additional Toolchains

The OSELAS.Toolchain-1.99.2 bundle comes with various predefined toolchains. Refer the `ptxconfigs/` folder for other definitions. To build additional toolchains we only have to clean our current toolchain projekt, removing the current `selected_ptxconfig` link and creating a new one.

```
~/OSELAS.Toolchain-1.99.2# ptxdist clean
~/OSELAS.Toolchain-1.99.2# rm selected_ptxconfig
~/OSELAS.Toolchain-1.99.2# ptxdist select
    ptxconfigs/any_another_toolchain_def.ptxconfig
~/OSELAS.Toolchain-1.99.2# ptxdist go
```

All toolchains will be installed side by side architecture dependend into directory

`/opt/OSELAS.Toolchain-1.99.2/architecture_part.`

Different toolchains for the same architecture will be installed side by side version dependend into directory

`/opt/OSELAS.Toolchain-1.99.2/architecture_part/version_part.`

## 4 Building Description

### 4.1 Prepare and Build

This chapter assumes we already installed PTXdist and the toolchain in a way the last chapter describes it.

It also assumes we have a big wire into the internet, as all the sources are downloaded from there. If we setup PTXdist to use a generic source archive directory the download only happens once.

Each ptxdist project uses a toolchain. This is the first thing we must setup prior starting the build.

```
jb@jupiter:~/ $ tar xf OSELAS.BSP-JB-Kritzel-trunk.tar.gz
jb@jupiter:~/ $ cd OSELAS.BSP-JB-Kritzel-trunk/Software
jb@jupiter:OSELAS.BSP-JB-Kritzel-trunk/Software $ ptxdist toolchain
/opt/OSELAS-1.99.2/avr/gcc-4.3.2-libc-1.6.2/bin
```

Now everything is ready to build the project:

```
jb@jupiter:OSELAS.BSP-JB-Kritzel-trunk/Software $ ptxdist go
```



Sometimes downloading a source archive may fail. Most of the time the maintainer moves the archive to another location or some delete the older releases when a new version was released. A simple solution is to search for this archive in the web, download it manually and store it into the generic source archive directory. If the archive file is already present, no download will happen.

---

When PTXdist has finished the Kritzel project, we can find all required parts in these directories and below:

- images/
  - shortcut.hex** For testing the CPU to USB FIFO connection
  - capturing.hex** The final firmware for Kritzel
  - generate\_pattern.hex** If you have a stk500v1 evaluationboard and a spare Atmega16 this firmware can generate test pattern
- local-host/bin/
  - avrdude** Main tool to flash everything in this project
  - ftdi\_eeeprom** Tool to setup USB-FIFO's EEPROM
  - find\_all** Tool to detect all connected FTDI USB-FIFO based devices
  - kritzel** Main tool to read data from kritzel
  - kritzel\_2vcd** Tool to convert Kritzel's native data format into VCD format
  - kritzel\_check** Verify a Kritzel data file
  - kritzel\_histogram** Tool to evaluate a Kritzel data file into a histogram
  - kritzel\_select** Tool to extract specific reports from a Kritzel data file

## **Part III**

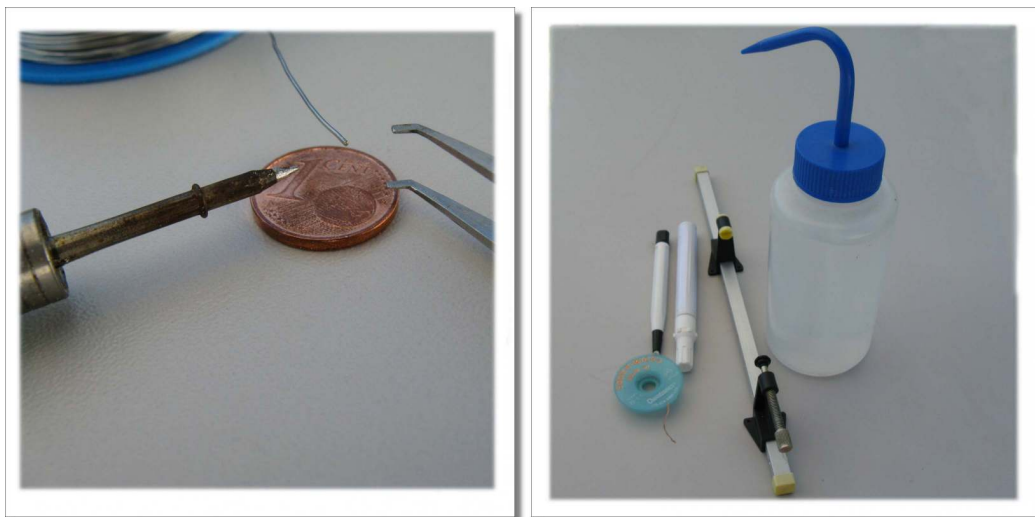
# **The hard Side of this Project**

## 5 Preparations and preconditions

### 5.1 Required Skills

To build this scanner you should have experience in working with SMT. Be aware the processor and the USB device are using a pitch of 0.67 mm. Soldering this kind of devices is not trivial. That's why you may need a huge amount of solder wick.

### 5.2 Required Tools



*Figure 5.1: Tools that makes this job easier (or harder?)*

These things and tools you should have:

- Some kind of magnifier (mine is a large one!)
- a very small pair of tweezers
- pointed tip soldering iron that is capable of soldering SMT
- thin solder (0.5 mm)
- solder wick (bulk pack!)
- something to chuck the card
- flux (optional)

- fibre glass pencil (optional)
- calm fingers (if not, additionally some bottles of beer)
- ohmmeter or continuity checker
- Linux based development host with USB and root permissions
- ISP device to program an 8bit AVR processor (Atmega16)
- light, light, light
- and a lot of patience

## 6 Bill of Material

### 6.1 Simple Bill for Assembling

Device	Value	Footprint	Quantity	Reference
TTL	74125	SO14	1	IC300
TTL	7414	SO14	1	IC301
I2C EEPROM	93C46	SO08	1	IC101
CAPACITOR	100n	0805	13	C102 C103 C104 C106 C108 C202 C203 C204 C205 C206 C300 C301 C302
CAPACITOR	10n	0805	1	C100
CAPACITOR	10p	0805	2	C200 C201
CAPACITOR	10 $\mu$ F	RM-2.5	1	C101
CAPACITOR	27p	0805	2	C105 C107
CAPACITOR	33n	0805	1	C109
CONNECTOR	HEADER6	2x3-RM2.54	1	X201
CONNECTOR	5PIN	1x5-RM2.54	1	X200
CPU	ATmega16	TQFP44	1	IC200
CRYSTAL	16MHz	HC49/S	1	XT200
CRYSTAL	6MHz	HC49/S	1	XT100
DIODE	BAV99	SOT23	9	D300 D301 D302 D303 D304 D305 D306 D307 D308
KEY	LSH-1301.9315	APEMTAST1	2	T200 T201
LED	GREEN	P-LCC-2	2	D1 D201
LED	RED	P-LCC-2	1	D200
LED	YELLOW	P-LCC-2	1	D202
REGULATOR	LM1117MPX-3.3	SOT223	1	IC302
RESISTOR	0R	0805	2	R319 R320
RESISTOR	1R	0805	1	R100
RESISTOR	22R	0805	8	R300 R301 R302 R303 R304 R305 R306 R307
RESISTOR	27R	0805	2	R101 R102
RESISTOR	100R	0805	6	R1 R204 R205 R206 R309 R310
RESISTOR	470R	0805	1	R104
RESISTOR	1k5	0805	1	R103
RESISTOR	2k2	0805	2	R105 R311

continued on next page

continued from previous page

Device	Value	Footprint	Quantity	Reference
RESISTOR	10k	0805	5	R106 R200 R201 R202 R203
RESISTOR	100k	0805	8	R308 R312 R313 R314 R315 R316 R317 R318
JACK	USB-B	USB-B	1	X100
USB	FT245BM	LQFP32	1	IC100
CASE	PP-2AA		1	
WIRE	flex. wire	white	160 cm	probe cable, 8 20 cm (signal)
WIRE	flex. wire	black	20 cm	probe cable, 1 20 cm (GND)
CRIMP	crimp contacts		9	for probe cable
TUBE	heat shrink tube		9 cm	for probe cable, 9 1 cm
CLIP	micro clip		10	for probe cable (expensive, but optional only!)
MARKER	Wire Marker		1	to mark probe 1
CABLE	USB Cable A-B		1	for Host to Kritzel connection
PCB	PCB		1	one of the main parts ;-)

## 6.2 Ordering Information

**Farnell** ([www.farnell.com](http://www.farnell.com)):

TTL: 7414, 74125,

CPU: Vendor: Atmel, Type: Atmega16

CRYSTAL: 6 MHz, 16 MHz

LEDs: Vendor: OSRAM, Type: TopLED

KEY: Vendor: Schurter, Type: LSH-1301.9315

REGULATOR: LM1117MPX-3.3, order no: "4660882"

**Reichelt** ([www.reichelt.de](http://www.reichelt.de)):

WIRE: AWG26 (0.14 mm<sup>2</sup>), Type: "LITZE WS" / "LITZE SW", white and black

TUBE: 2:1, 2.4 mm, order no: "SDH 2,4 SW"

**RS** (<http://de.rs-online.com/web/>):

CRIMP contacts: 0.64 mm, order no: "532-456"

REGULATOR: LM1117MPX-3.3, order no: "2509474047"

**Segor** (Berlin, [www.segor.de](http://www.segor.de)):

I2C EEPROM: Vendor Atmel, Type: 93C46

CRYSTAL: 6 MHz, 16 MHz

USB: Vendor: FTDI, Type: FT245BM

JACK: Vendor: Yamichi, Type USB-B-001

CAPACITOR: all

RESISTOR: all

DIODE: all

CONNECTOR: all

CABLE: USB cable A-B

**Conrad** ([www.conrad.de](http://www.conrad.de)):

CASE: PAC TEC, PP-2AA ([www.pactecenclosures.com](http://www.pactecenclosures.com))

Ordering: Conrad Electronic, No. 523150 - 62 ("KUNSTSTOFFGEHAEUSE 94X63X28")

Note: I found this case on their German website only.

**Digikey**

REGULATOR: "LM1117MPX-3.3\*"

**Micro-Clip:**

<http://www.amrhein-online.de/messtechnik/produkte/mikro/microclip.php?lang=en>

**BetaLayout ([www.pcb-pool.com](http://www.pcb-pool.com)):**

Circuit Board (55.1 mm    82.54 mm    1.6 mm / 2.17 in    3.25 in    0.063 in)

# 7 Assembly

## 7.1 The PCB from the CAD system

This project contains the PCB layout file in EAGLE \*.brd format. Figure 7.1 shows the CAD output of the board. It's a hand routed layout, so it was possible to get by with two layers only.

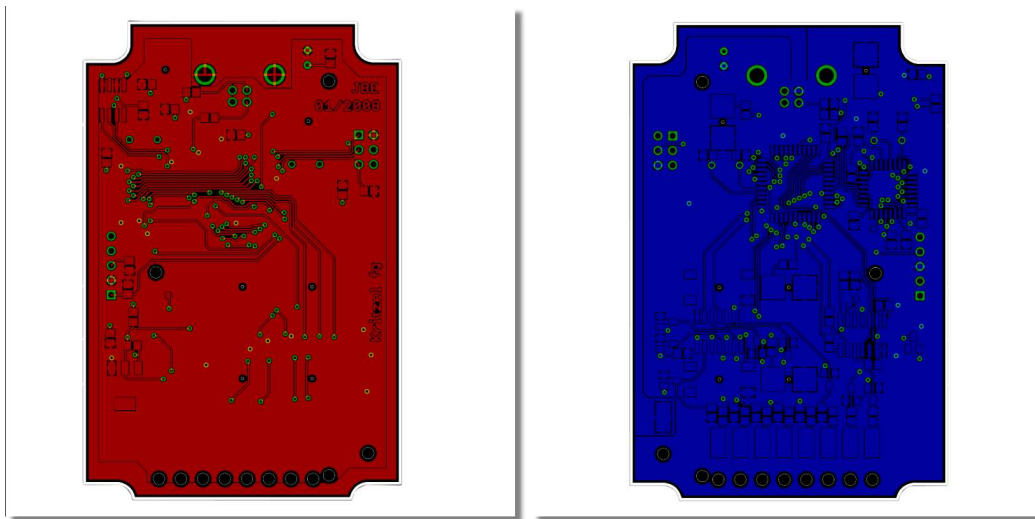


Figure 7.1: The board from CAD (left top, right bottom layer)

## 7.2 The PCB from the Manufacturer

Figure 7.2 shows the result after production. It was done in a five working day job.

## 7.3 Check the plain PCB

First of all, check if the board has some shorts between power supply (+5V) and GND. If not, continue with the next step. If yes, hmmm, try to find and solve it first.

Best place to check for shorts between Power and GND is the capacitor C101. Don't measure at the USB-B connector! The R100 is not soldered yet, so it makes no sense to measure at this connector. As C101 is of a regular THD type (through-hole device), it will be easy to do the measuring at both sides of the PCB.

Do also a visual check of the whole PCB. Sometimes cracks can be detected in this way.

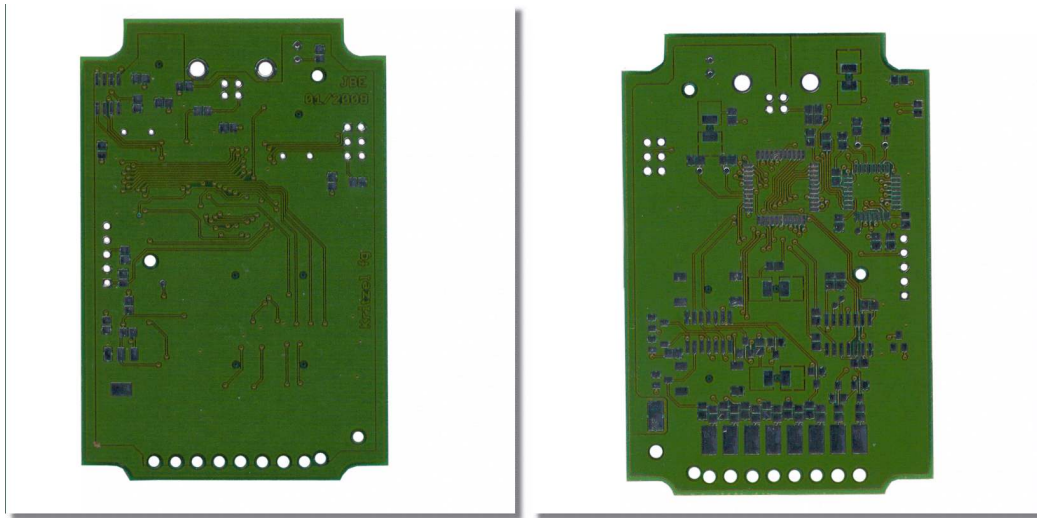


Figure 7.2: The fresh board (left top, right bottom layer)

## 7.4 Soldering

Switch on the soldering iron, we will start now!



Don't solder all components in one step! Follow the steps in this manual, as all steps are done with a check at their end. This will simplify searching for any kind of failure. And to do it step by step will ensure you can build an accurate housing.

For the component placement on the PCB refer figure 8.1 (top side) and 8.2 (bottom side) on page 51.

### 7.4.1 Step 1: The small devices

First step is to solder the smallest devices: Resistors, capacitors and the 3.3 voltage regulator as shown in figure 7.3:

In Germany many people call these kind of small devices "Huehnerfutter" (chicken feed).



Do not solder the LEDs and the buttons to the board now. There is an additional step to be done, before soldering these devices!

### 7.4.2 Checking Step 1

Once again check at the capacitor C101 if you made a short between +5V and GND. It could be a good idea soldering a few devices, doing a short test, soldering the next few devices and doing the next short test (and so on). As the resistor R100 should be soldered, you can also do the measuring at the USB-B connector now.

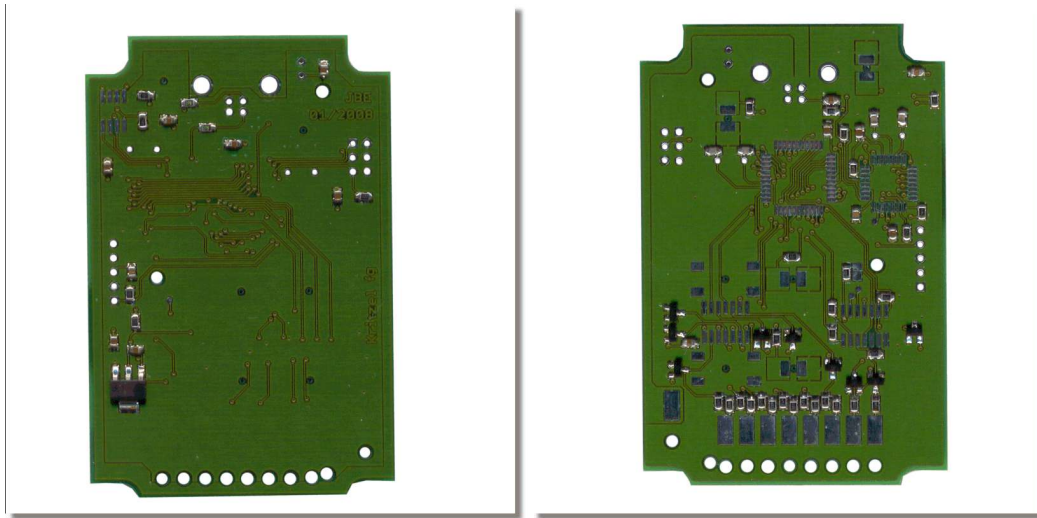


Figure 7.3: The first step: "Das Hühnerfutter" ;-)

### 7.4.3 Step 2: The Processor

In this step we solder the CPU and some other devices on demand to program it. This is the time, where you need calm fingers, a very thin soldering iron and the thin solder. If you miss one or all of these, its time to use the solder wick now.

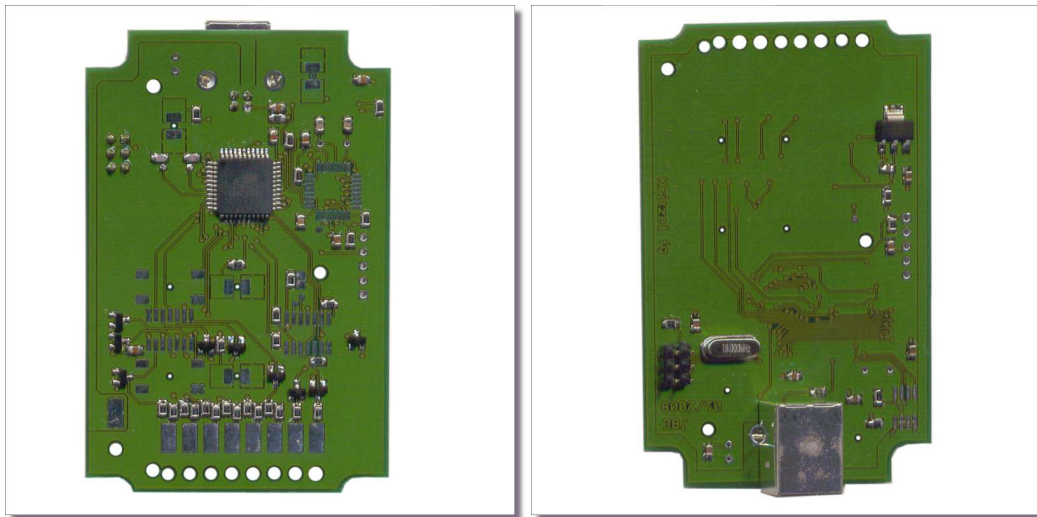


Figure 7.4: The second step: CPU and some other devices

The 6 pin ISP connector (X201) is always required. Also the crystal (XT200). The USB-B plug (X100) only if your ISP is not able to supply the CPU.

### 7.4.4 Checking Step 2

Time to start with the ISP. First the connection diagram for the ISP connector (X201) used on the Kritzel PCB. The pin 1 of this connector is marked with a rectangle pad on the PCB as shown in figure 7.5.

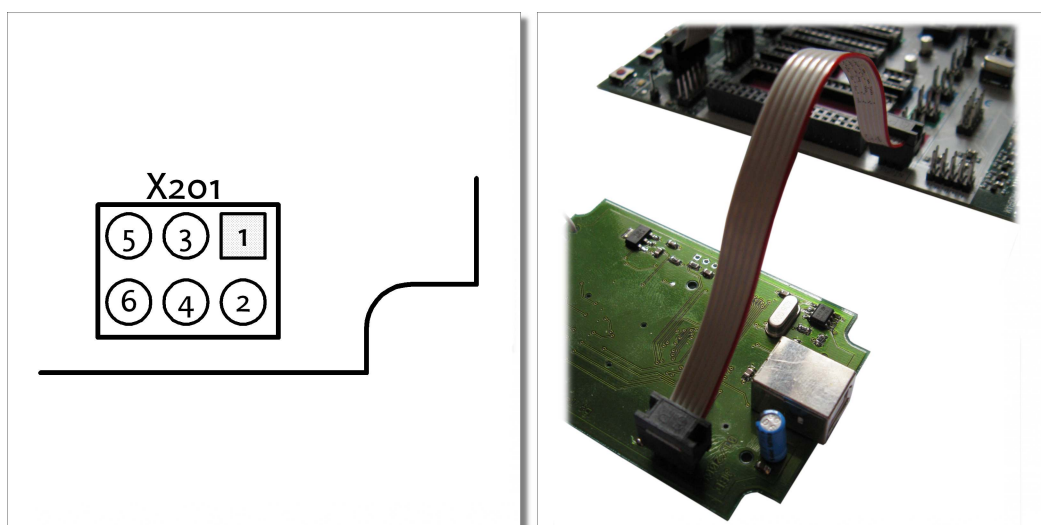


Figure 7.5: ISP Pinning (left) and how to connect to the STK500 (right)

The signal directions listed in table 7.1 are from the sight of Kritzel.

Pin No	Function	Function	Pin No
1	MISO (output)	VCC (bidirectional)	2
3	SCK (input)	MOSI (input)	4
5	RES# (input)	GND (passive)	6

Table 7.1: ISP connection

Note: The VCC pin can be used to supply the ISP from Kritzel or to supply the Kritzel from the ISP.

#### 7.4.4.1 Test if the CPU is working:

Connect the ISP cable to the Kritzel board through X201. If you are using the STK500 (like me) you can supply the whole Kritzel from the STK500. There is no need to solder the USB B to supply the board at this time. In this case close the VTARGET jumper on the STK500. If you already soldered the USB B plug, do not connect Kritzel to your host via USB, when the VTARGET jumper on the STK500 is closed.

A first and simple test to check if everything is working is to try to get an ISP connection to the processor. You can do it with:

```
# local-host/bin/avrdude -p atmega16 -c stk500v1 -t -v
```

Note: I'm using the stk500 to program my devices. That's why I'm using the '-c stk500v1' option in all shown commands here. If you are using a different programmer, change this option.

This will start the debug console to gain access to the whole processor. If it stops with a failure, first check if the RES# line could be manipulated by the ISP programmer. If not, increase the pull up resistor. 2k2 are way to small for the stk500 kit. 10k are working with the stk500, but there are potential ISP programmers in the wild that need a higher resistor value than 10k (maybe 47k).

When there is no failure, the ISP seems to get in contact with the processor. In this case it should output the processor's signature and current fuse settings:

```
avrdude: Version 5.4, compiled on Nov  4 2007 at 13:07:47
```

```
Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
```

```
System wide configuration file is "/etc/avrdude.conf"
```

```
User configuration file is "/home/.avrduderc"
```

```
User configuration file does not exist or is not a regular file, skipping
```

```
Using Port          : /dev/ttyS0
Using Programmer    : stk500v1
AVR Part           : ATMEGA16
Chip Erase delay    : 9000 us
PAGEL              : PD7
BS2                : PA0
RESET disposition   : dedicated
RETRY pulse         : SCK
serial program mode : yes
parallel program mode : yes
Timeout            : 200
StabDelay          : 100
CmdexeDelay         : 25
SyncLoops          : 32
ByteDelay          : 0
PollIndex          : 3
PollValue          : 0x53
Memory Detail      :
```

Memory	Type	Mode	Delay	Block Size	Poll Indx	Paged	Size	Page Size	#Pages	MinW	MaxW	Polled ReadBack
eeeprom		4	10	128	0	no	512	4	0	9000	9000	0xff 0xff
flash		33	6	128	0	yes	16384	128	128	4500	4500	0xff 0xff
lock		0	0	0	0	no	1	0	0	9000	9000	0x00 0x00
lfuse		0	0	0	0	no	1	0	0	9000	9000	0x00 0x00
hfuse		0	0	0	0	no	1	0	0	9000	9000	0x00 0x00
signature		0	0	0	0	no	3	0	0	0	0	0x00 0x00
calibration		0	0	0	0	no	4	0	0	0	0	0x00 0x00

```
Programmer Type : STK500
```

```
Description      : Atmel STK500 Version 1.x firmware
```

```
Hardware Version: 2
Firmware Version: 1.15
Vtarget          : 5.1 V
Varef            : 5.1 V
Oscillator       : Off
SCK period       : 5.5 us
```

avrdude: AVR device initialized and ready to accept instructions

Reading | ##### | 100% 0.03s

```
avrdude: Device signature = 0x1e9403
avrdude: safemode: lfuse reads as E1
avrdude: safemode: hfuse reads as 99
avrdude>
```

You can leave this mode by entering quit.

The ATmega processor is controlled via fuses. Do not omit this step, it's mandatory. The fuse configuration we need in this application:

- External Crystal/Resonator High Freq. Startup time 1k CK + oms
- brown out detection disabled
- (brown out detection level 2.7V)
- Boot reset vector not programmed
- Boot flash size 1kiB
- do not preserve EEPROM content while chip erase
- CKOPT fuse programmed (as we are using a crystal above 8MHz!)
- SPI based ISP enabled
- JTAG disabled
- On Chip Debug disabled

To activate this configuration run the following command:

```
# local-host/bin/avrdude -p atmega16 -c stk500v1 -U hfuse:w:0xc9:m -U lfuse:w:0xee:m
```

### 7.4.5 Step 3: The USB device

After the CPU seems to run, its time to solder the USB feature. This means the FT245 (IC100) and its crystal (XT100). Its also the time now to solder the USB B (X100) plug if not already done.

Sorry, no pictures available for this step. You can refer to figure [7.10](#) for the fully soldered PCB.

### 7.4.6 Checking Step 3

Again check if the power supply is not shorted out. Now we are at the exciting moment of connecting Kritzel via USB to our host.

To get some help on what happens (or maybe not), the following tools should be available on your host:

- `dmesg` Command to see kernel messages
- `ftdi_sio` Kernel driver for the FTDI USB device family
- `lsusb` Command to get information about connected USB devices
- Your favoured terminal program (like `minicom`, `microcom`, `nanocom`)

If your Linux distribution does not support autoload of kernel modules, you must load the `ftdi_sio` manually. This must be done as root:

```
# sudo modprobe ftdi_sio
```

Plug in Kritzel's USB cable into your host now.

If it runs well, the kernel will output something like the text below if it detects the FTDI USB device. You can see this output when you run the `dmesg` command.

```
[...]
usb 1-2: new full speed USB device using ohci_hcd and address 3
usb 1-2: configuration #1 chosen from 1 choice
ftdi_sio 1-2:1.0: FTDI USB Serial Device converter detected
drivers/usb/serial/ftdi_sio.c: Detected FT232BM
usb 1-2: FTDI USB Serial Device converter now attached to ttyUSB0
[...]
```

Note: Don't ask me why the driver detects the FT245 as an FT232BM...

Kritzel is now accessible via device node `/dev/ttyUSB0`. Or another `USB<number>`. `<number>` depends on your system and how many other such devices are currently connected.

Now we will check the whole communication. This project contains a testware called `echo` for Kritzel. It echos all chars from the input without modification to the output.

Again connect Kritzel to your ISP (consider that only one power supply is active, the ISP or the USB host!) and program the testware into it:

```
# local-host/bin/avrdude -p atmega16 -c stk500v1 -U flash:w:images/echo.hex:i
```

Disconnect the ISP and/or connect Kritzel again to your USB host. Run your favoured terminal program now. I'm using `nanocom` here as an example.

```
# nanocom /dev/ttyUSB0 -b 9600 -p n -s 1 -d 8 -f n -e n
```

Press CTRL+T for menu options

```
*****Line Status*****
9600 bps      8 Data Bits    n Parity      1 Stop Bits    n Flow Control n echo
*****
```

Note: Replace `/dev/ttyUSB0` with the device node your Kritzel registered at in your system.

The serial line settings like baudrate aso. are not required here. But without them, `nanocom` will output error messages.

All characters you enter must occur immediately on the screen. And they must occur as the same characters you type! If they do so, Kritzel is working as expected.

If nothing happens at the steps above, check if Kritzel is getting the 5V supply from your host via USB. One of my PCBs was broken and shorted the AVCC signal (R104, pin 2 to IC100, pin 30) to GND. The USB device seemed dead. It became alive again after fixing this short.

If something happens, but you don't receive the exact characters you type, one or more data bits might be broken between the ATmega16 and the FT245.

### 7.4.7 Step 4: The Input/Output Drivers

All right, near to completion. Solder IC300 and IC301 now. After that the probe cables must be soldered.

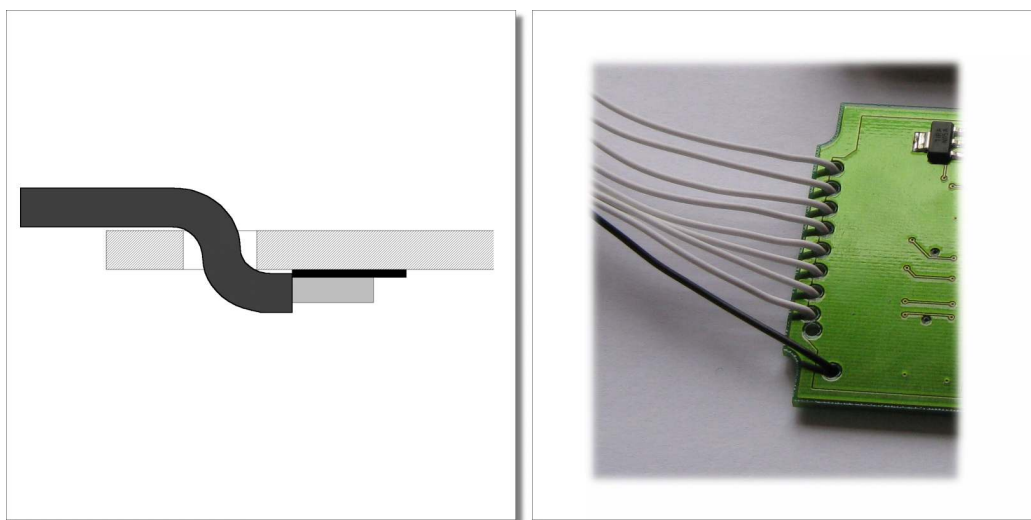


Figure 7.6: The forth step: Probe cables

The hole in the PCB should work as a strain relief. So use it in a way shown in figure 7.6 in the left picture. At the end it should look like the picture on the right side.

Use different colours for the probes. At least for GND and the single bidirectional probe channel. This will simplify the usage later on.

### 7.4.8 Checking Step 4

For this test we flash the final firmware into Kritzel. So - once again - connect the ISP to Kritzel and run:

```
# local-host/bin/avrdude -p atmega16 -c stk500v1 -U flash:w:images/capturing.hex:i
```

By activating the stimulation port we can check if the event recognition works as expected. To do so, short the stimulation output (probe 1) with one of the other event input ports one after each other.

1. Short stimulation probe 1 with event probe 2
2. Run: `local-host/bin/kritzel -r -s 500 -i 1000000 -a 500000 /dev/ttyUSB0`

We are connected to:

Type: kritzel\_fg

Rev.: 0.1.5

Using scan intervall of 500 ns

Note: Kritzel does not realize requested stimulation!

Using stimulation intervall of 1024000 ns, length of 500000 ns and level of 1

Scanning...

Wait a few seconds, then terminate it by entering CTRL+C

3. Next run `local-host/bin/kritzel_histogram -p 2 test_file.krt` on this dataset:

Reading data from file test\_file.krt

# Histogram for cycle measuring

# Measuring

# Reporting interval: 500 ns

# Of signal signal\_2 (2)

# Data sets over all: 49229. Counted cycles: 12118. Erroneous cycles: 0.

1020000	4
1020500	6031
1026500	1545
1027000	4538

The result should look like this. Note: Kritzel is not perfect, so you will never get only one exact result in the histogram.

Repeat the third step on the other probes. They should not contain any events in this case. If they do, there seems to be a short on the PCB.

Now connect probe 3 ... 7 to the stimulation probe 1 one after each other and repeat the three checking steps shown above.

### 7.4.9 Finishing the housing

The LEDs and the buttons are not soldered yet. This is important as we need the small holes in the PCB that mark the center of these devices as a reference where to drill the top case and to drill the probe holes. Only with not soldered LEDs and buttons we can insert the card into the top case.

In this step we are drilling the case. Refer to figure 7.7 and figure 7.8. Start with marking the holes we need for the 9 probe cables. Insert the card into the top case and mark with a pencil where the 9 probe cables should pass the case.

A triangular file will help to mark these holes to drill them to the right size in the next step.

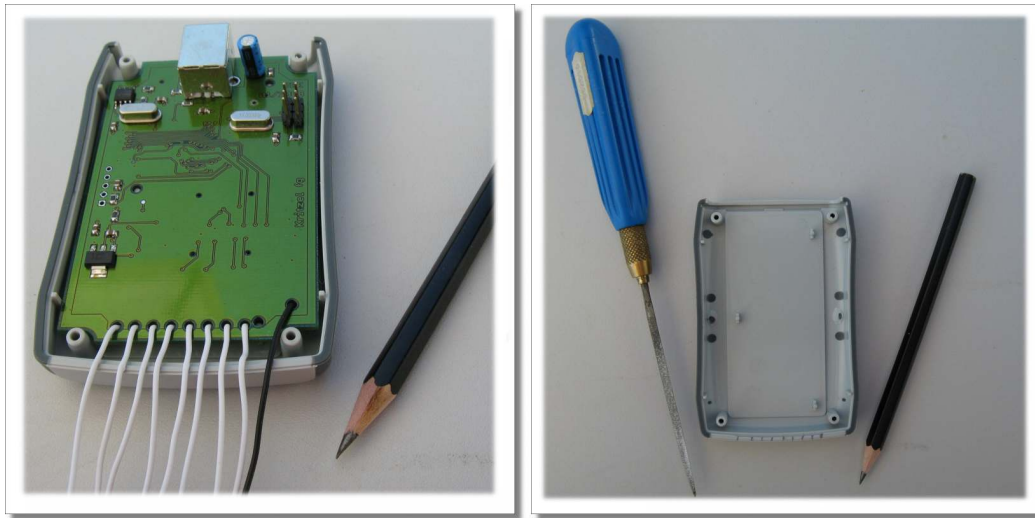


Figure 7.7: How to mark and drill the probe holes

Assemble top and bottom case to finish this step. Now it's easy and safe to drill these marked holes with a drill of the size of your probe cables (figure 7.8, right picture).



Figure 7.8: Finishing the holes for the probe cables

After drilling the probe holes, we can drill the LED and button holes. Refer to figure 7.9 for this step. Insert the PCB into the top case again and drill with an 0.8 mm drill the six holes like shown in the left side of figure 7.9.

This step is the reason, why you shouldn't have soldered the LEDs and buttons yet!

When drilling is finished, you got a top case like shown in figure 7.9 on the right side. The six red circles mark the six small holes.

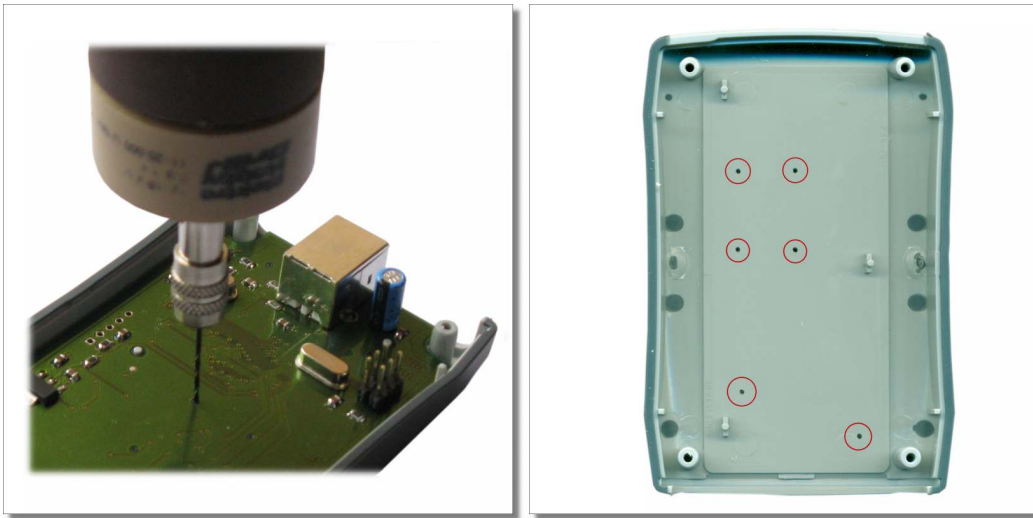


Figure 7.9: How to mark and drill the correct holes

The next step is to enlarge these holes as shown in figure 7.10.

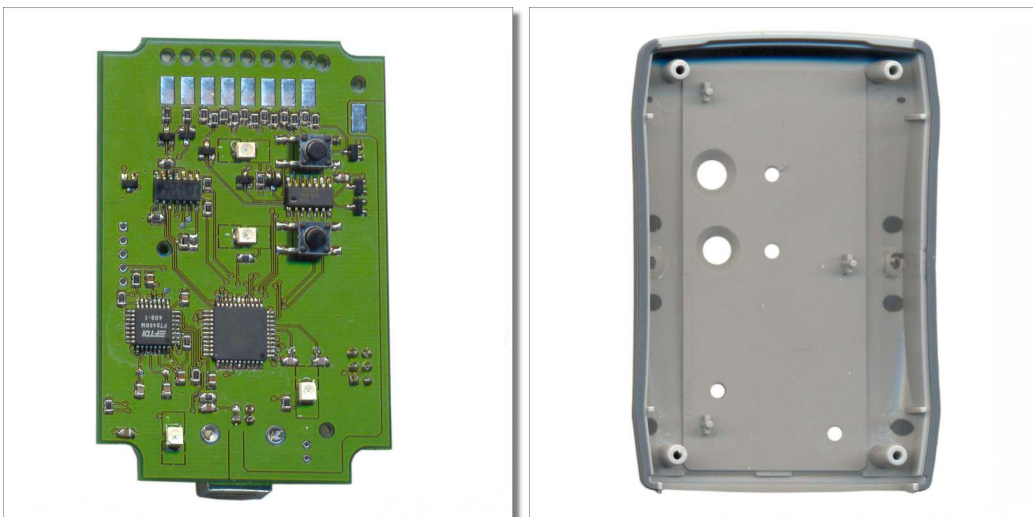


Figure 7.10: The PCB with LEDs and buttons (left) and finished holes (right)

Holes for the LEDs should be 1 mm ... 2 mm in size. The base hole for the buttons must be at least 4 mm in size. But in addition they need a counterbore as the button's main body is too high to fit between the PCB and the top case.

The right picture in figure 7.11 shows the required hole in the panel for the USB connector. The size of this hole is 12.5 mm width and 11.5 mm height. It must be located exactly in the center of the bare panel.

After everything is done in this section, you can solder the LEDs and the two buttons and you are done.

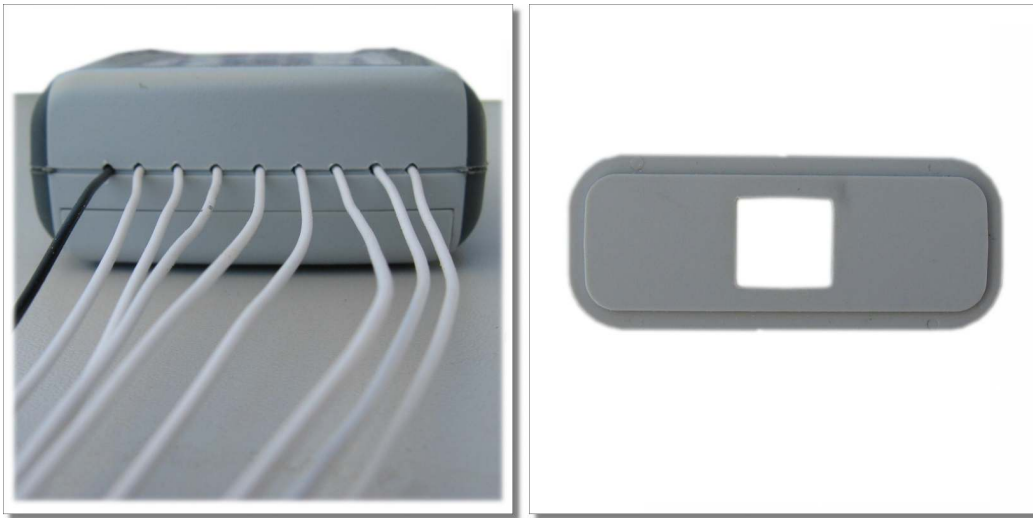


Figure 7.11: Finished probes (left) and the USB panel (right)

## 7.5 Programming the FTDI's EEPROM

The FTDI device uses an external EEPROM. It's not programmed yet, so this will be the next step to do.

Use the `find_all` utility to see if your FTDI is visible to the system.

```
# local-host/bin/find_all
```

```
Number of FTDI devices found: 1
```

```
Checking device: 0
```

```
Manufacturer: FTDI, Description: USB <-> Serial
```

Create this type of configuration file for the `ftdi_eeprom` utility and name it `ftdi.config`:

```
vendor_id=0x0403      # Vendor ID
product_id=0x6001     # Product ID

max_power=50          # Max. power consumption: value * 2 mA.
                      # Use 0 if self_powered = true.

#####
# Strings #
#####
manufacturer="JBE"    # Manufacturer
product="Logic Scanner" # Product name
serial="01-01"         # Serial number
```

```
#####
# Options #
#####
self_powered=false      # Turn this off for bus powered
remote_wakeup=false     # Turn this on for remote wakeup feature
use_serial=true         # Use the serial number string

# Normally out don't have to change one of these flags
BM_type_chip=true       # Newer chips are all BM type
in_is_isochronous=false # In Endpoint is Isochronous
out_is_isochronous=false # Out Endpoint is Isochronous
suspend_pull_downs=false # Enable suspend pull downs for lower power
change_usb_version=true  # Change USB Version
usb_version=0x0200       # Only used when change_usb_version is enabled

#####
# Misc #
#####

filename="eeprom.new"   # Filename, leave empty to skip file writing
```

What does the entries in this file mean:

**product\_id** Keep this at 0x6001, as this will ensure the Linux ftdi\_sio will claim this device. Otherwise you will need your own kernel driver.

**self\_powered** Kritzal is not self powered. So this entry must be **false**.

**manufacturer** Enter your name here or something like this.

**product** Enter the product name or description here.

**serial** Insert here something unique per device. If its unique you can use it in udev rules to do something special on a per device base.

**use\_serial** Should be **true**. Otherwise this device seems to have a bug that prevents the kernel driver to identify this device as a BM type.

**remote\_wakeup** This project does not use this feature. So keep it **false**.

**BM\_type\_chip** The value **true** ensures, the libftdi writes the correct **0x4000** as the *version* of this chip. The Linux ftdi\_sio driver uses this value to determine if this is an AM or BM device. Note: This should be **true** only if you are really using a BM type!

**in\_is\_isochronous** don't know yet

**out\_is\_isochronous** don't know yet

**change\_usb\_version** Can be **true**, if you want to be USB 2.0 compliant. Note: The FTDI device is still a full speed device only, even if you change it to USB 2!

**usb\_version** If **change\_usb\_version** is **true**, this defines the USB version to be changed to.

When the EEPROM does not contain any data, the chip works as expected. In this case some of the info from the running `lsusb -v` command contains predefined values only.

Here are the predefined values. If everything went correctly, you should see this output:

Bus 001 Device 003: ID 0403:6001 Future Technology Devices International, Ltd 8-bit FIFO

Device Descriptor:

bLength	18
bDescriptorType	1
bcdUSB	1.10
bDeviceClass	0 (Defined at Interface level)
bDeviceSubClass	0
bDeviceProtocol	0
bMaxPacketSize0	8
idVendor	0x0403 Future Technology Devices International, Ltd
idProduct	0x6001 8-bit FIFO
bcdDevice	4.00
iManufacturer	1 FTDI
iProduct	2 USB <-> Serial
iSerial	0
bNumConfigurations	1

Configuration Descriptor:

bLength	9
bDescriptorType	2
wTotalLength	32
bNumInterfaces	1
bConfigurationValue	1
iConfiguration	0
bmAttributes	0x80
(Bus Powered)	
MaxPower	90mA

Interface Descriptor:

bLength	9
bDescriptorType	4
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	2
bInterfaceClass	255 Vendor Specific Class
bInterfaceSubClass	255 Vendor Specific Subclass
bInterfaceProtocol	255 Vendor Specific Protocol
iInterface	2 USB <-> Serial

Endpoint Descriptor:

bLength	7
bDescriptorType	5
bEndpointAddress	0x81 EP 1 IN
bmAttributes	2
Transfer Type	Bulk
Synch Type	None
Usage Type	Data
wMaxPacketSize	0x0040 1x 64 bytes

```
    bInterval          0
Endpoint Descriptor:
    bLength            7
    bDescriptorType     5
    bEndpointAddress   0x02  EP 2 OUT
    bmAttributes       2
        Transfer Type   Bulk
        Synch Type      None
        Usage Type      Data
    wMaxPacketSize     0x0040  1x 64 bytes
    bInterval          0
Device Status:        0x0000
    (Bus Powered)
```

And now - while the scanner is attached (check with the `find_all` utility) - run:

**FIXME:** *Why are root permissions required to do the next step?*

```
# local-host/bin/ftdi_eeeprom --flash-eeeprom ftdi.config
```

This command will disconnect the scanner. You must unplug the scanner and plug it in again to let it occur again.

After plug in you can check the result of the EEPROM programming step with:

```
# local-host/bin/find_all
```

```
Number of FTDI devices found: 1
Checking device: 0
Manufacturer: JBE, Description: Logic Scanner
```

```
# lsusb -v
```

Now some values are changed. The output should look like this now:

```
Bus 001 Device 006: ID 0403:6001 Future Technology Devices International, Ltd 8-bit FIFO
Device Descriptor:
  bLength            18
  bDescriptorType     1
  bcdUSB             1.10
  bDeviceClass        0 (Defined at Interface level)
  bDeviceSubClass     0
  bDeviceProtocol     0
  bMaxPacketSize0     8
  idVendor            0x0403 Future Technology Devices International, Ltd
  idProduct           0x6001 8-bit FIFO
  bcdDevice           4.00
  iManufacturer       1 JBE
```

```
iProduct          2 Logic Scanner
iSerial           3 01-01
bNumConfigurations 1
Configuration Descriptor:
  bLength          9
  bDescriptorType   2
  wTotalLength     32
  bNumInterfaces    1
  bConfigurationValue 1
  iConfiguration    0
  bmAttributes      0x80
    (Bus Powered)
  MaxPower         100mA
Interface Descriptor:
  bLength          9
  bDescriptorType   4
  bInterfaceNumber  0
  bAlternateSetting 0
  bNumEndpoints     2
  bInterfaceClass   255 Vendor Specific Class
  bInterfaceSubClass 255 Vendor Specific Subclass
  bInterfaceProtocol 255 Vendor Specific Protocol
  iInterface        2 Logic Scanner
Endpoint Descriptor:
  bLength          7
  bDescriptorType   5
  bEndpointAddress  0x81 EP 1 IN
  bmAttributes      2
    Transfer Type    Bulk
    Synch Type       None
    Usage Type       Data
  wMaxPacketSize    0x0040 1x 64 bytes
  bInterval         0
Endpoint Descriptor:
  bLength          7
  bDescriptorType   5
  bEndpointAddress  0x02 EP 2 OUT
  bmAttributes      2
    Transfer Type    Bulk
    Synch Type       None
    Usage Type       Data
  wMaxPacketSize    0x0040 1x 64 bytes
  bInterval         0
Device Status:      0x0000
  (Bus Powered)
```

Note: This PTXdist project contains a patched version of the *ftdi library*. Without this patch the *bmAttributes*

contains an illegal value, so the `lsusb -v` command complains about it. If you are not using the tools and libraries from this PTXdist project ensure you are using at least the *libftdi-0.12* (up to 0.11 this failure exists).

Tip: If you are going to build more than one scanner device, use different serial numbers in each device. With this feature you can create special *udev* rules ensuring fixed device node names on a serial number base.

## 7.6 Front Panel

For a front panel you can use my template I made for my devices. Figure 7.12 shows an enlarged panel. The title photo shows this panel in use.

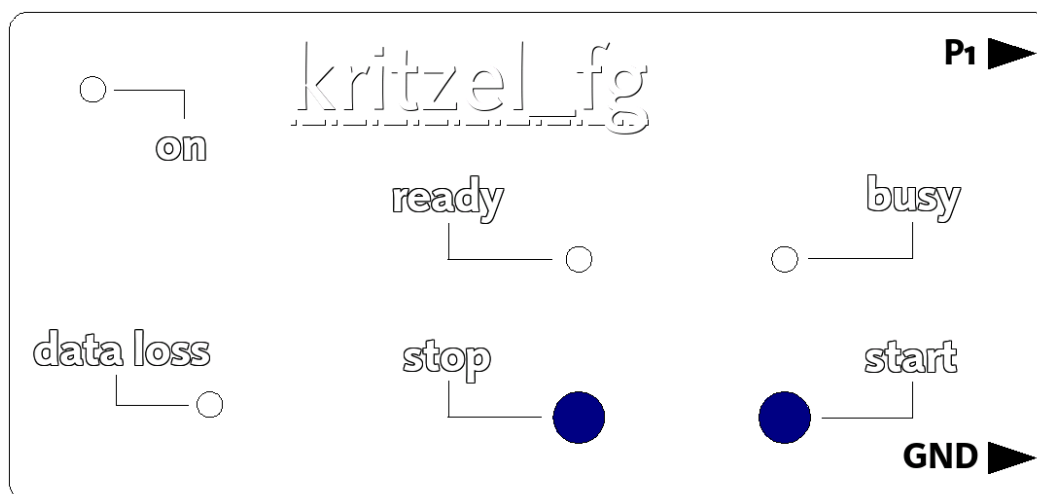


Figure 7.12: Front Panel Template

### 7.6.1 Panel's Dimensions

Note: Notation here is (x,y), unit is mm.

Lets assume the position (0.0,0.0) is at the bottom left corner. Then the top right corner is at (81.0,38.0). All corners must be rounded off. The other controls are:

**on LED** is at position (6.5,32.0), needs a hole of 1..2 mm in size

**data loss LED** is at position (15.6,7.5), needs a hole of 1..2 mm in size

**ready LED** is at position (44.3,18.8), needs a hole of 1..2 mm in size

**stop button** is at position (44.3,6.5)

**busy LED** is at position (60.3,18.8), needs a hole of 1..2 mm in size

**start button** is at position (60.3,6.5)

I printed the panel on some kind of photo paper, to get a perfect look. With a ticket-punch I made the LED holes. To inhibit any blur, I'm using an adhesive film at its top. And a two side adhesive film to stick the front panel onto the top case.

## 8 Layout Component Placement

## 8.1 Top Side Component Placement

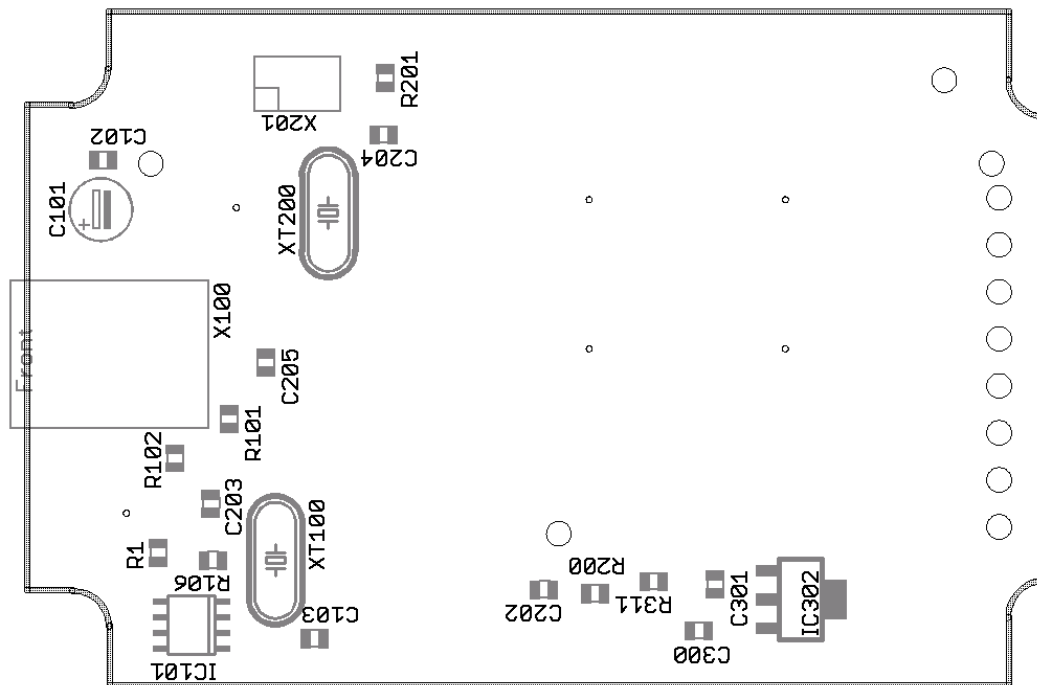


Figure 8.1: Top side component placement



## 9 Schematics

The schematics are on the following four pages.

Page 54 shows the block diagram. It shows the main three parts of this system and their interconnection:

- Block diagram
- Main CPU
- USB connection
- Signal detection

Page 55 shows the Atmega16 CPU that does all the work in this system.

This CPU runs at 5 V power supply to get the highest available speed for it. An ISP connector allows in system programming and updating the firmware. But everytime this must be done, the case must be opened.

This page also shows X200. It was intended to debug the Kritzel, but it was never used. It was easier to test and debug various subroutines on the host, prior their usage at the target.

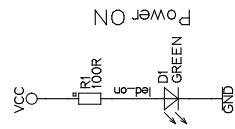
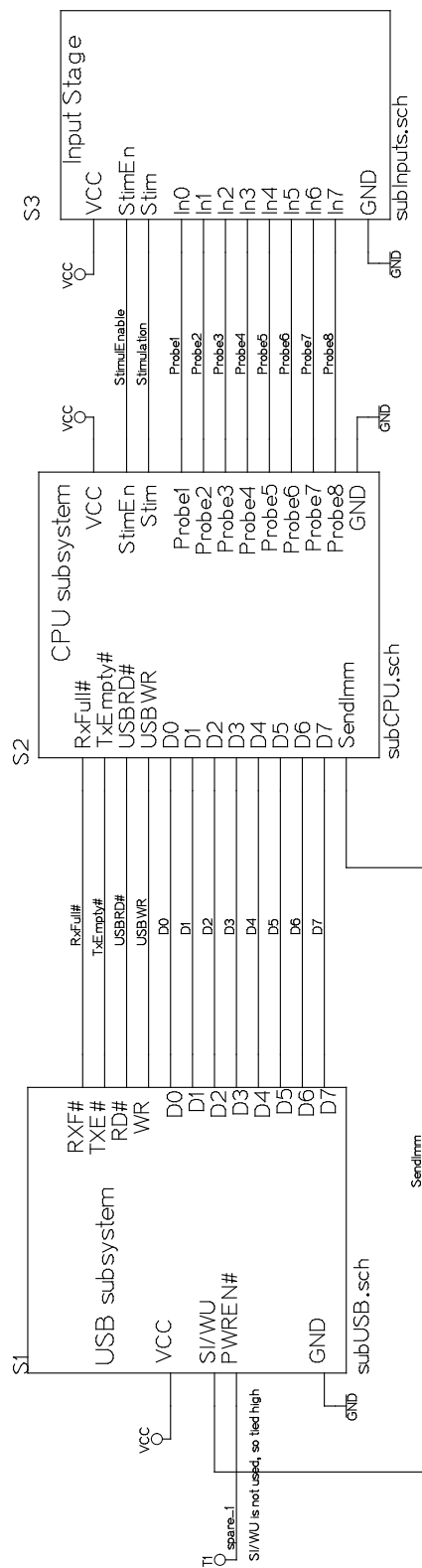
Page 56 shows the USB connection.

USB connection is done by an parallel to USB converter. This device has internal FIFOs for full speed data transfers and a really ugly CPU interface. Don't trust the write FIFO busy line! After each written byte the FTD245 rises its busy line and lowers it again when there is still space in the FIFO. But: The rising edge after each byte occurs about 500 ns after the byte was written! So to send the next byte you must wait about 500 ns to get a **valid** busy signal.

The EEPROM shown on this page do not need any programming prior soldering. It can be later done from host's side via USB. Kritzel works without a programmed EEPROM, but I recommend to program it. At least the serial number is useful, as it provides a rule for udev to generate a fix name whenever this device gets attached to the host. Without it, its hard to distinguish between real RS232 to USB converter and Kritzel.

Page 57 shows signal detection and generation.

Nothing really exciting here. Some ESD protection, some level shifter. Channel 1 and 2 are of type 3.3 V (but 5 V tolerant), all other channels are using 5 V for their supply voltage. And they are negated! This will be inverted later in software. Using a 7414 here was a bad idea, as it's not trivial to invert 6 bits per byte in assembler. It needs one register to load prior the invert operation (so it needs two instructions). Wastes some CPU cycles. The ESD protection for channel 1 is also used for its 5 V tolerance. Disadvantage here is, even if the output is tri-state, if the CPU continues to send PWM signals the 3.3 V power supply shows ripples.



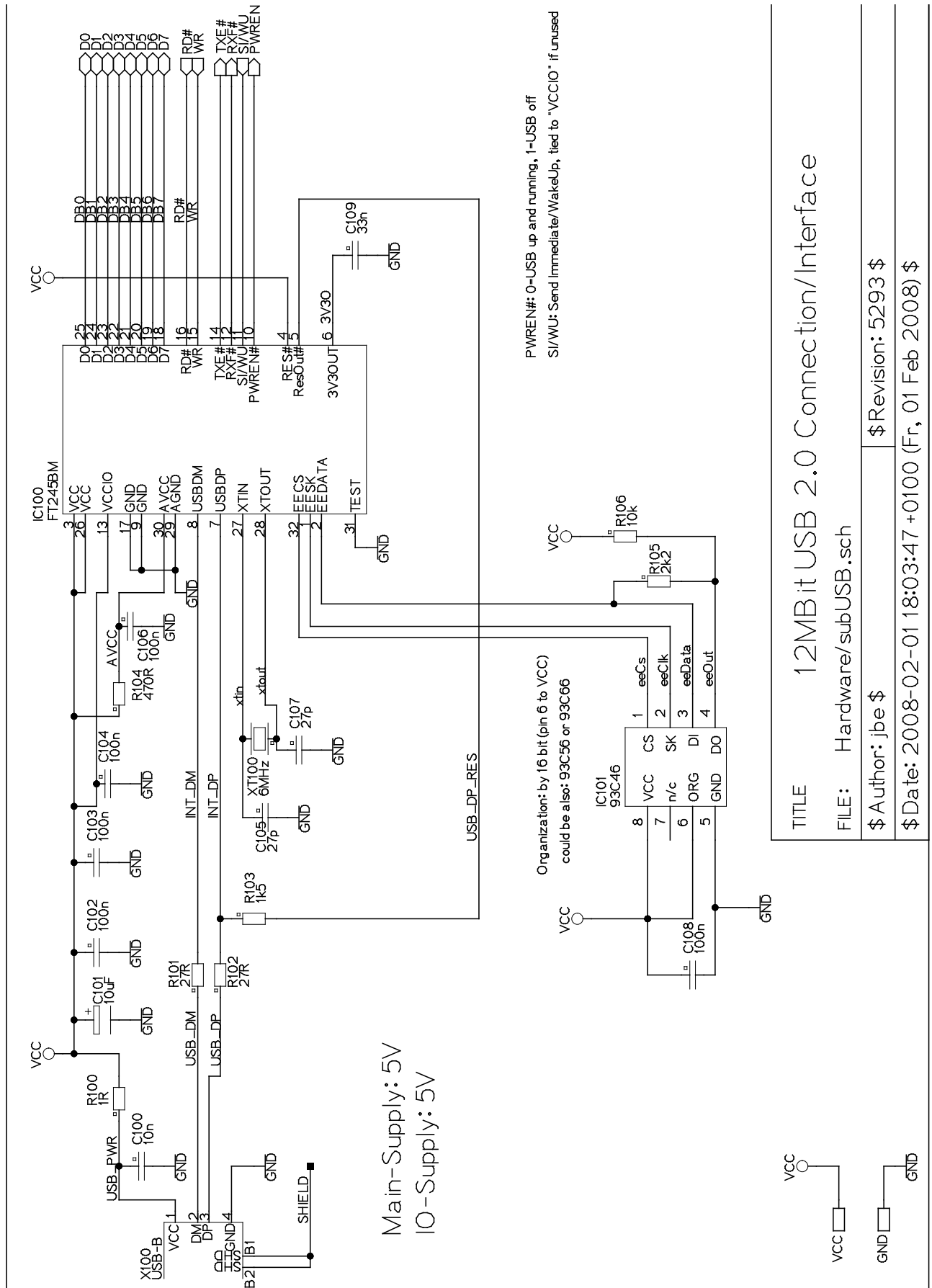
TITLE Kritzel scanner: Main Page	
FILE: Hardware/MainPage.sch	
\$ Author: jbe \$	\$Revision: 5293 \$
\$Date: 2008-02-01 18:03:47 +0100 (Fr, 01 Feb 2008) \$	

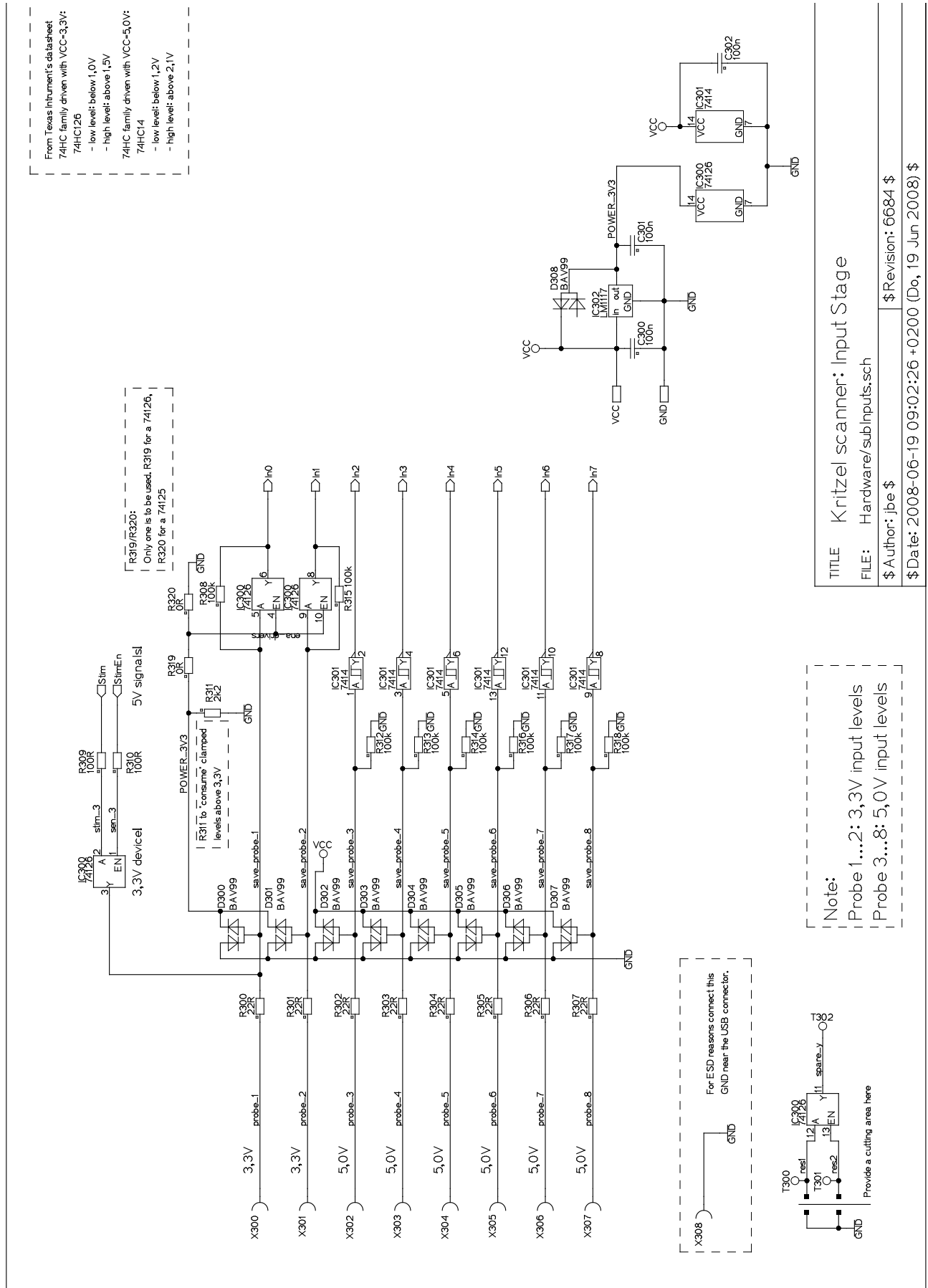


FILE: Hardware/subCPU.sch

\$ Author: jbe \$ \$ Revision: 6684 \$

\$Date: 2008-06-19 09:02:26 +0200 (Do, 19 Jun 2008) \$





## 10 Help

In case of any question you can contact me at: [projects@kreuzholzen.de](mailto:projects@kreuzholzen.de)